UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

GABRIEL JUNGES BARATTO

COMPARAÇÃO ENTRE OS MODELOS PRÉ-TREINADOS GPT-3 E BERT NA ESTIMATIVA DE ESFORÇO DE SOFTWARE POR ANALOGIA A PARTIR DE REQUISITOS TEXTUAIS

PATO BRANCO

2022

GABRIEL JUNGES BARATTO

COMPARAÇÃO ENTRE OS MODELOS PRÉ-TREINADOS GPT-3 E BERT NA ESTIMATIVA DE ESFORÇO DE SOFTWARE POR ANALOGIA A PARTIR DE REQUISITOS TEXTUAIS

Comparison between the pre-trained models GPT-3 and BERT in software effort estimation by analogy from text requeriments

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia da Computação do Curso de Bacharelado em Engenharia da Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Profª. Drª. Eliane Maria de Bortoli Fávero

PATO BRANCO

2022

GABRIEL JUNGES BARATTO

COMPARAÇÃO ENTRE OS MODELOS PRÉ-TREINADOS GPT-3 E BERT NA ESTIMATIVA DE ESFORÇO DE SOFTWARE POR ANALOGIA A PARTIR DE REQUISITOS TEXTUAIS

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia da Computação do Curso de Bacharelado em Engenharia da Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 09/dezembro/2022

Eliane Maria de Bortoli Fávero
Doutor
Universidade Tecnológica Federal do Paraná

Dalcimar Casanova
Doutor
Universidade Tecnológica Federal do Paraná

Jefferson Tales Oliva
Doutor
Universidade Tecnológica Federal do Paraná

PATO BRANCO
2022

RESUMO

A estimativa do esforço de *software* necessário para o desenvolvimento dos requisitos de usuário é uma das etapas mais importantes do ciclo de desenvolvimento de *software*. Diversas são as técnicas para realizar essa atividade, mas a Inteligência Artificial, com métodos de Aprendizado de Máquina associados ao Processamento de Linguagem Natural (PLN), podem contribuir, automatizando parte desse processo de estimativa, com base em textos de requisitos (ex. histórias de usuário) de projetos passados, trata-se da Estimativa de Esforço de Software por Analogia (EESA). A EESA requer conhecimento acerca de dados históricos de projetos e, por esse motivo, raramente é usada durante a fase de planejamento do processo de desenvolvimento. Ainda, normalmente na fase inicial, o que se tem são requisitos em formato textual. Sendo assim, esse trabalho objetivou realizar a comparação dos modelos de representação textual GPT-3 e BERT quando aplicados à EESA a partir da representação textual de requisitos de usuário. Para isso, esse trabalho aplicou o modelo pré-treinado contextualizado GPT-3 na representação de características textuais para a inferência de EESA e em seguida, realizou a comparação de seu desempenho com o modelo SE^{3}M – *Software Effort Estimation Embedding Model* –, que fez uso do modelo de linguagem pré-treinado BERT. Para o pré-processamento dos requisitos, utilizaram-se expressões regulares, e para a inferência das estimativas, foram treinadas redes neurais *feedforward*. Para a avaliação do modelo, dois experimentos com diferentes particionamentos do corpus de texto foram realizados, dos quais foram extraídas as métricas Erro Quadrático Médio (MSE), Erro Absoluto Médio (MAE) e Mediana dos Erros Absolutos (MdAE). Obtidas as métricas dos dois experimentos, compararam-se as mesmas com os resultados do modelo SE^{3}M. Das métricas obtidas, destaca-se o MAE de 3,80$\pm$1,20 (intervalo de confiança de 95%), que provém do experimento que utilizou um particionamento entre-repositórios. Este resultado é semelhante ao obtido pelos modelos SE^{3}M e Deep-SE (também comparado), porém com a vantagem do modelo GPT-3 ter sido utilizado sem ajuste-fino. A principal limitação levantada na utilização do GPT-3 foi o custo, que é exigido tanto para extração da representação textual quanto para o ajuste-fino.

Palavras-chave: estimativa de esforço de software; estimativa de esforço de *software* por analogia; *deep learning*; *zero-shot learning*; processamento de linguagem natural (pln).

ABSTRACT

Estimating the software effort required to develop each of user requirements is one of the most important steps of the software development cicle. There are several techniques to perform this activity, but Artificial Intelligence, such as Machine Learning methods associated with Natural Language Processing (NLP), can contribute by automating part of this estimation process, based on requirements texts (e.g. user stories). Analogy-based software effort estimation (ASEE) requires knowledge about historical project data and is rarely used during the planning phase of the development process. Still, normally in the initial phase, it is common to have requirements in textual format. Therefore, this work aimed to perform a comparation of the text representation models GPT-3 and BERT when applied on the ASEE from the analysis of textual user requirements. For that, this work applied the contextualized pre-trained model GPT-3 in the representation of textual features for the inference of software effort estimates by analogy and then compared its performance with that of the SE^{3}M – Software Effort Estimation Embedding Model –, which uses the pre-trained language model BERT. For the pre-processing of the requirements, regular expressions were used, and for the inference of the estimates, feedforward neural networks were trained. To evaluate the model, two experiments with different partitioning of the text corpus were performed, from which the metrics Mean Squared Error (MSE), Mean Absolute Error (MAE) and Median Absolute Error (MdAE) were extracted. Once the metrics of the two experiments were obtained, they were compared with the results of the SE^{3}M model. In the end, a better performance was observed from the model that used the GPT-3 compared to the one that used the BERT model (SE^{3}M). Of the metrics obtained, the MAE of 3.80$\pm$1.20 (95% confidence interval) stands out, which comes from the experiment that used a cross-repository partitioning. This result is similar to that obtained by the SE^{3}M and Deep-SE (also compared) models, but with the advantage that the GPT-3 model was used without fine-tuning. The main limitation raised in using GPT-3 was the cost, which is required both for extracting the text representation and for fine-tuning.

Keywords: story point estimation; analogy-based effort estimation; deep learning; zero-shot learning; natural language processing (nlp).

LISTA DE FIGURAS

Figura 1 – Exemplos de plot 3D de word vectors. **22**

Figura 2 – Matriz de co-ocorrência para a sentença "o gato sentou no tapete". . . **24**

Figura 3 – Exemplo de uso do mecanismo de auto-atenção. **27**

Figura 4 – Tendência de tamanhos de modelos de PLN de última geração com o tempo. **29**

Figura 5 – Comparação entre auto-atenção e auto-atenção mascarada. **29**

Figura 6 – Estrutura de uma FNN. **32**

Figura 7 – Ilustração do problema do *vanishing gradient* **34**

Figura 8 – Ilustração de um bloco de memória LSTM com uma célula **35**

Figura 9 – Diferença entre arquiteturas (a) *Encoder-Decoder* tradicional (RNN/LSTM) e (b) *Attention*. **37**

Figura 10 – Exemplo de particionamento *k-fold* com um k igual a 5. . . . . . **40**

Figura 11 – O conceito de validação cruzada aninhada *k-fold* **40**

Figura 12 – Metodologia proposta **48**

Figura 13 – Lista de caracteres e termos removidos do texto dos requisitos (*stoplist*). Em azul destacam-se os caracteres especiais e os dígitos, enquanto em laranja destacam-se as *tags html* removidos. **49**

Figura 14 – Histograma representando a medida do esforço em relação à sua frequência no *corp_ES*. **52**

Figura 15 – Histograma representando o número de palavras em cada sentença no *corp_ES*. **57**

Figura 16 – Histograma representando o número de palavras em cada sentença no *corp_ES*, limitado a 500 palavras. **57**

LISTA DE TABELAS

Tabela 1 – Número de requisitos textuais (histórias de usuário) e descrição dos projetos utilizados nos experimentos **51**

Tabela 2 – Avaliação dos resultados obtidos paras os experimentos E1 e E2. Para ambas as métricas avaliadas, quanto menor for o valor, melhor o resultado. **59**

Tabela 3 – Valores do MAE obtidos em cada subconjunto no experimento E2 e do experimento análogo realizado no SE^{3}M, com dados dos respectivos projetos utilizados para avaliação do modelo. Também são apresentados o identificador do projeto (ID), o número de requisitos por projeto (Requisitos/projeto), a média (M_E) e o desvio padrão (DP) do esforço por requisito em cada projeto. Quanto menor o valor do MAE, melhor o resultado. Os menores valores de MAE de cada projeto foram destacados em negrito. **59**

Tabela 4 – Erro Absoluto Médio (MAE), Erro Quadrático Médio (MSE) e Mediana dos Erros Absolutos (MdAE) obtidos ao aplicar o modelo GPT-3 (Z-SE^2), comparados aos modelos SE^{3}M (BERT) e Deep-SE (Word2Vec). Para todas as métricas, quanto menor o valor melhor o resultados. **60**

LISTA DE QUADROS

Quadro 1 – Utilização de texto para estimativa de esforço/tipo de análise e técnica de representação textual. **43**
Quadro 2 – Tecnologias utilizadas para o trabalho **46**
Quadro 3 – Especificações do computador utilizado para treinamento do modelo de inferência **47**
Quadro 4 – Arquitetura da rede neural do modelo de inferência. **53**
Quadro 5 – Hiperparâmetros testados pelo *grid search* **53**
Quadro 6 – 5 amostras de requisitos da base *corp_ES* (ainda sem pré-processamento) **55**
Quadro 7 – 5 amostras de requisitos da base *corp_ES* após pré-processamento . **56**
Quadro 8 – Representação por *word embeddings* obtidos para requisitos do *corp_ES*. Cada vetor exibido corresponde ao *word embedding* obtido para um requisito textual, e contém 2048 características (valores em decimal). **58**

LISTA DE ABREVIATURAS, SIGLAS E ACRÔNIMOS

Siglas

AM	Aprendizado de Máquina
ANN	Redes Neurais Artificiais, do Inglês *Artificial Neural Networks*
API	Interface de Programação de Aplicação, do Inglês *Application Programming Interface*
DL	*Deep Learning*
EES	Estimativa de Esforço de Software
EESA	Estimativa de Esforço de Software por Analogia
FNN	Redes Neurais *Feedforward*, do Inglês *Feedforward Neural Networks*
GPT	*Transformer* Pré-treinado Generativo, do inglês *Generative Pre-trained Transformer*
HPO	Otimização de Hiperparâmetros, do Inglês *Hiper-Parameter Optimization*
HTML	Linguagem de marcação de hipertexto, do Inglês *HyperText Markup Language*
IA	Inteligência Artificial
IC	Intervalo de Confiança
LSTM	Memória Longa de Curto Prazo, do Inglês *Long Short-Term Memory*
MdAE	Erro Absoluto Mediano, do Inglês *Median Absolute Error*
MLP	Perceptron de Múltiplas Camadas, do Inglês *Multilayer Perceptron*
MSE	Erro Quadrático Médio, do Inglês *Mean Squared Error*
NLTK	*Natural Language Toolkit*
OPT	*Transformer* Pré-Treinado Aberto, do Inglês *Open Pre-trained Transformer*
PLN	Processamento de Linguagem Natural
RHN	*Recurrent Highway Networks*
RNN	Redes Neurais Recorrentes, do Inglês *Recurrent Neural Networks*

SE^3M	Modelo de Embedding para Estimativa do Esforço de Software, do Inglês *Software Effort Estimation Embedding Model*
TF	Frequência do Termo, do Inglês *Term Frequency*
TF-IDF	*Term Frequency-Inverse Document Frequency*
UML	Linguagem de Modelagem Unificada, do Inglês *Unified Modeling Language*
Z-SE^2	*Zero-shot Software Effort Estimator*

Acrônimos

BERT	*Bidirectional Encoder Representations from Transformers*
BOW	*Bag-of-words*, ou saco de palavras na tradução literal do Inglês
CBOW	BOW Contínuo, do inglês *Continuous Bag of Words*
GloVe	*Global Vectors*
MAE	Erro Absoluto Médio, do Inglês *Mean Absolute Error*
POS	*Part-of-Speech*
ReLU	Unidade Linear Retificada, do Inglês *Rectified Linear Unit*
Word2Vec	*Word to vector*

SUMÁRIO

1 INTRODUÇÃO 11

1.1 Considerações Iniciais 11

1.2 Objetivos 13

1.2.1 Objetivo Geral 13

1.2.2 Objetivos Específicos 13

1.3 Justificativa 14

1.4 Estrutura do trabalho 15

2 REFERENCIAL TEÓRICO 16

2.1 Modelos de Estimativa de Esforço de Software (EES) 16

2.1.1 Estimativa de Esforço de Software por Analogia (EESA) 17

2.2 Processamento de Linguagem Natural (PLN) 18

2.3 Pré-processamento de Textos 19

2.4 Representação Textual 20

2.4.1 Modelos clássicos de representação de palavras 20

2.4.2 *Word embeddings* 22

2.4.2.1 *Word embeddings* sem contexto 23

2.4.2.2 *Word embeddings* contextualizados 24

2.4.3 *Transfer Learning* 24

2.4.4 Modelos de Representação Pré-treinados 25

2.4.4.1 BERT 26

2.4.4.2 Família GPT 27

2.5 Algoritmos de aprendizado e métricas 29

2.5.1 Aprendizado de Máquina 30

2.5.2 Redes Neurais Artificiais (ANNs) 31

2.5.2.1 Funções de Ativação 32

2.5.2.2 *Deep Learning* 33

2.5.2.3 Redes Neurais Recorrentes (RNNs) 33

2.5.2.4 Mecanismo de Atenção 36

2.5.3 Métricas de Avaliação 37

2.5.3.1 Intervalos de Confiança 38

2.5.4 Métodos de Validação e Particionamento de Dados 39
2.5.5 Otimização de Hiperparâmetros 41
2.6 Trabalhos Relacionados 41
3 MATERIAIS E MÉTODO 45
3.1 Materiais 45
3.2 Método 48
3.2.1 Coleta de dados e Pré-processamento 49
3.2.2 Extração de características/representação textual 50
3.2.3 Métodos de Particionamento 50
3.2.4 Método de Inferência 51
3.2.5 Avaliação 53
4 RESULTADOS 55
4.1 Preparação do Dataset 55
4.2 Representação Textual via GPT-3 56
4.3 Métricas obtidas 58
4.4 Discussão dos resultados 60
4.4.1 Possíveis Limitações e Melhorias 62
5 CONCLUSÃO 64
REFERÊNCIAS 66
GLOSSÁRIO 75
ANEXO A ARTIGO - REVISÃO SISTEMÁTICA DA LITERATURA 77

1 INTRODUÇÃO

Este capítulo fornece uma visão geral sobre o tema, bem como o contexto deste projeto, de forma a apresentar o objetivo geral e os específicos a que se propõe. Para isso, é realizada uma revisão de literaturas e trabalhos relacionados à área, fornecendo assim justificativa e fundamentação teórica para o mesmo.

1.1 Considerações Iniciais

Estimar o esforço de software é o processo de aproximar o número de horas necessárias para concluir tarefas individuais, bem como os materiais, pessoas, equipamentos e suprimentos necessários para realização delas (PMI, 2008). Ela é uma das atividades que compõem a etapa de planejamento do processo de desenvolvimento de software.

Apesar de complexa, a Estimativa de Esforço de Software (EES) é uma das tarefas mais importantes da engenharia de software, uma vez que imprecisões em sua definição podem ocasionar aumentos no tempo e orçamento de projetos, comprometendo a qualidade final do software e a satisfação do cliente (TRENDOWICZ; JEFFERY, 2014).

Um método de estimativa que vem sendo amplamente utilizado atualmente é a Estimativa de Esforço de Software por Analogia (EESA), o que trata-se de um método não paramétrico que utiliza dados históricos de outros projetos para realizar as estimativas para o projeto corrente (CHIU; HUANG, 2007; PMI, 2021).

De acordo com Chiu e Huang (2007), a EESA consiste em identificar um ou mais projetos históricos, semelhantes ao projeto a ser desenvolvido e, em seguida, derivar as estimativas dele. O diferencial deste método está em utilizar estimativas passadas na previsão de novas estimativas. Isso permite aplicações de métodos de Aprendizado de Máquina (AM), possibilitando a previsão de estimativas por sistemas inteligentes, facilitando o trabalho de equipes de desenvolvimento de software.

Isso é ressaltado pelos autores Idri, Amazal e Abran (2015), que classificam a técnica por analogia como uma técnica de AM. Os mesmos autores destacam ainda, que os modelos de AM também têm obtido uma atenção significativa de pesquisadores no campo da estimativa do esforço de software, pois podem modelar a relação complexa entre esforço e atributos de software (fatores de custo), especialmente quando essa relação não é linear e não parece ter qualquer forma predeterminada.

Sendo assim, ao realizar EESA, relacionando estimativas de esforço e requisitos textuais de projetos semelhantes do passado, pode-se dizer que a estimativa de esforço passa a fazer parte dos problemas de Processamento de Linguagem Natural (PLN), podendo assim fazer uso de suas soluções mais recentes.

O PLN é um ramo da Inteligência Artificial (IA) que vem crescendo muito nos últimos anos. De acordo com Lauriola, Lavelli e Aiolli (2022), PLN é uma área repleta de tarefas com-

plexas, sofisticadas e desafiadoras relacionadas ao idioma, como tradução automática, resposta a perguntas, resumo e assim por diante. Segundo eles, o PLN envolve o projeto e a implementação de modelos, sistemas e algoritmos para resolver problemas práticos de compreensão das linguagens humanas, geralmente com o uso de sistemas de computação.

Dentre esta classe de problemas, encontra-se a tarefa de extrair características textuais de uma sentença, de modo que o texto seja convertido para um formato que os modelos de AM consigam trabalhar. As técnicas capazes de realizarem esta tarefa são denominadas métodos de representação textual. Ainda, destas técnicas, os métodos de *word embedding* compõem o estado-da-arte, sendo utilizados para diversas tarefas.

Os métodos de *word embedding* (neste trabalho chamado apenas de *embeddings*) objetivam principalmente capturar a semântica de uma determinada palavra em um contexto específico. Esse método permite que as palavras sejam representadas de forma densa e com baixa dimensionalidade, facilitando tarefas de AM que usam características textuais.

Os modelos de *embedding* pode ser classificados em contextualizados e não contextualizados. Os modelos não contextuais não se atentam à posição das palavras em uma frase e desconsideram as várias implicações que elas podem ter (BARUA *et al.*, 2021), ou seja, não consideram todas as palavras de uma sentença para definir o contexto de cada palavra. Para exemplificar, os autores citam a palavra "banco", a qual possui várias interpretações, dependendo do contexto da frase e que, em um modelo não contextualizado, é considerado um único contexto, mesmo que essa palavra ocorra muitas vezes no texto, em diferentes situações.

Os métodos de *word embedding* contextualizados para representação textual permitem a extração do contexto de cada palavra em um texto (DEVLIN *et al.*, 2018). Esse método difere do original (não contextualizado), pois produz diferentes representações vetoriais para a mesma palavra em um texto, o que varia de acordo com o seu contexto e, portanto, é capaz de capturar semânticas contextuais de palavras ambíguas (AKBIK; BERGMANN; VOLLGRAF, 2019), além de tratar as questões de polissemia.

Fazendo uso dessa forma de representação textual, alguns trabalhos (IONESCU, 2017; CHOETKIERTIKUL *et al.*, 2019; FÁVERO; CASANOVA; PIMENTEL, 2022) vêm implementando métodos para EESA nos últimos anos.

Dentro desse novo paradigma de *embeddings*, o modelo *Bidirectional Encoder Representations from Transformers* (BERT) (DEVLIN *et al.*, 2018) tem apresentado os melhores resultados em tarefas de PLN e, portanto, vem sendo amplamente utilizado em diversas aplicações. Sua utilização tem ocorrido por meio de modelos de linguagem pré-treinados e disponibilizados por seus autores (ex. BERT_base e BERT_large (DEVLIN *et al.*, 2018)). Fávero, Casanova e Pimentel (2022) aplicou BERT para inferir estimativa de esforço a partir de textos de requisitos de software e obteve bons resultados.

O GPT-3 (*Transformer* Pré-treinado Generativo, do inglês *Generative Pre-trained Transformer*) (BROWN *et al.*, 2020) é uma evolução do modelo GPT-2, fornecido pela empresa OpenAI, o qual disponibiliza modelos pré-treinados contextualizados de forma similar ao BERT, mas

possui um número muito maior de parâmetros, sendo considerado o estado-da-arte. Sendo assim, a proposta deste projeto é aplicar o modelo pré-treinado contextualizado GPT-3 ao mesmo problema de EESA relatado por Fávero, Casanova e Pimentel (2022), em que foi aplicado o BERT, com e sem ajuste-fino[1]. Com isso, pretende-se avaliar possíveis diferenças de desempenho do método de representação textual utilizado, justificando vantagens e desvantagens entre ambos os modelos.

A partir desse comparativo procurou-se responder a seguinte questão de pesquisa: **O modelo pré-treinado GPT-3 é tão eficiente quanto o modelo BERT para a EESA a partir de requisitos textuais?**

Uma hipótese para esta pergunta de pesquisa é que o modelo GPT-3 sem ajuste fino apresente um desempenho igual ou melhor ao modelo BERT ajustado. Esta hipótese baseou-se no fato de o modelo GPT-3 *Babbage* ter sido treinado em um volume muito maior de dados textuais e apresentar um número de parâmetros dezenas de vezes maior, quando comparado ao modelo BERT.

1.2 Objetivos

Nesta sessão são apresentados os objetivos dessa proposta, que são divididos em objetivo geral e objetivos específicos, os quais representam respectivamente o resultado esperado para esse trabalho e os passos necessários para a sua realização.

1.2.1 Objetivo Geral

- Comparar o desempenho da aplicação dos modelos pré-treinados contextualizados GPT-3 e BERT (empregado no SE^{3}M) na representação de características textuais para a inferência de EESA.

1.2.2 Objetivos Específicos

A partir do objetivo geral deste trabalho, foram levantados os seguintes objetivos específicos:

- Realizar a preparação da base de dados para a aplicação de modelos pré-treinados em modelos de AM;

[1] Ajuste-fino, do Inglês *fine-tuning*, é o processo de retreinar uma rede neural que já havia sido treinada, de modo a prepará-la para a utilização em uma tarefa específica. Geralmente, esta técnica é utilizada em redes neurais pré-treinadas em um grande volume de dados, de modo a reaproveitar todo o seu conhecimento para a tarefa alvo. (YIN *et al.*, 2017).

- Gerar a representação textual por meio de *embeddings*, fazendo uso do modelo pré-treinado contextualizado GPT-3;
- Aplicar métodos de AM para inferir EESA com base em representações obtidas via modelos pré-treinados contextualizados;
- Realizar um comparativo das métricas obtidas via modelo de inferência de estimativas com base no modelo pré-treinado GPT-3 e BERT (modelo SE^{3}M);

1.3 Justificativa

A primeira das vantagens de se utilizar modelos de linguagem pré-treinados com base em uma quantidade massiva de dados, está no menor volume de treinamento e no menor esforço necessário para a construção da arquitetura do modelo. A segunda vantagem está na precisão melhorada de suas predições, pois considerando o volume de dados de entrada utilizados em seu treinamento, o aprendizado adquirido por esses modelos é proporcionalmente maior (HAN *et al.*, 2021).

Além de ser um modelo de linguagem pré-treinado, o GPT-3 (assim como o BERT) utiliza-se do mecanismo de atenção, o que proporciona uma contextualização mais real das palavras de uma dada sentença, melhorando sua precisão. A aplicação de técnicas mais aprimoradas de análise de contexto em modelos de linguagem pré-treinados pode melhorar consideravelmente a precisão em uma vasta gama de aplicações de tarefas que envolvem PLN, definindo um novo patamar para o estado-da-arte (FLORIDI; CHIRIATTI, 2020).

Essa recente combinação das técnicas de *embeddings* contextualizados abre espaço para um novo nível de previsão em aplicações que já faziam uso de modelos pré-treinados. Uma dessas aplicações é a própria tarefa de EESA, que já vinha se utilizando modelos de linguagem para o cálculo de estimativas. Desta forma, um modelo pré-treinado contextualizado (o estado-da-arte em representação textual) foi aplicado na proposta desse projeto de conclusão de curso, com o qual esperava-se verificar sua utilidade para a EESA. Destaca-se que o acadêmico, o qual construiu um conhecimento ímpar em técnicas de PLN e AM, tornando-o mais competitivo no mercado. Além disso, os resultados da implementação deste projeto trouxeram contribuições importantes para a área de pesquisa de PLN aplicada à engenharia de software.

Desta forma, a proposta deste trabalho visou preencher essa lacuna de pesquisa, realizando a aplicação métodos de AM no modelo de linguagem GPT-3, de modo a extrair as suas métricas de desempenho na inferência da EESA, e compará-las com as do modelo SE^{3}M (FÁVERO; CASANOVA; PIMENTEL, 2022), que utilizou o modelo de linguagem BERT.

1.4 Estrutura do trabalho

Esse trabalho se encontra organizado da seguinte forma: o Capítulo 2 apresenta o referencial teórico necessário ao desenvolvimento dos objetivos desse trabalho. Durante o Capítulo 3 é descrito como foi realizado este trabalho, expondo os materiais e o método utilizados. No Capítulo 4 são apresentados os resultados obtidos com a realização de cada um dos objetivos específicos propostos, bem como uma discussão sobre estes resultados. E por fim, no Capítulo 5, são apresentadas as considerações finais sobre o trabalho.

2 REFERENCIAL TEÓRICO

Este capítulo realiza uma revisão dos principais conceitos que permeiam o desenvolvimento do trabalho proposto, buscando na literatura os diversos autores com conhecimento sobre o tema. Ainda neste capítulo, é realizado um levantamento de trabalhos de pesquisa semelhantes encontrados na literatura e uma sondagem do atual estado da arte. Os conceitos relevantes que envolvem o trabalho proposto dividem-se em Modelos de Estimativa de Esforço de Software (EES), Processamento de Linguagem Natural (PLN), Pré-Processamento de Textos, Representação Textual, e Algoritmos de Aprendizado e Métricas de Avaliação de Desempenho.

2.1 Modelos de Estimativa de Esforço de Software (EES)

A Estimativa de Esforço de Software (EES) é uma das etapas do desenvolvimento de software. Para entendê-la, primeiro é necessário revisar o conceito de esforço. O esforço é a medida de quantas unidades de trabalho serão gastas para completar uma determinada tarefa ou atividade (ex. uma funcionalidade de um software), o que pode variar, sendo atribuídas como homem/dia, homem/hora, entre outros. Quando se fala em custo de software, refere-se ao valor monetário correspondente às unidades de medida gastas para o seu desenvolvimento. (ABDUKALYKOV, 2011).

Diversas classificações para modelos de Estimativa de Esforço de Software (EES) foram propostas nas últimas décadas, as quais apresentam pequenas diferenças conforme o ponto de vista de cada autor. Algumas delas são apresentadas na sequência.

Os modelos de EES podem ser subdivididos em algorítmicos/paramétricos e não-algorítmicos. Os primeiros são aqueles que se utilizam de modelos matemáticos ou algorítmicos aplicados a atributos do projeto para calcular a sua estimativa. São exemplos desses modelos *Construtive Cost Model* II (COCOMO II) (BOEHM; ABTS; CHULANI, 2000), ou Modelo de Custo Construtivo em tradução literal do Inglês, e a Análise de Pontos de Função (CHOI; PARK; SUGUMARAN, 2012). Já os não-algorítmicos são aqueles baseados em técnicas de Aprendizado de Máquina (AM), as quais dependem de dados históricos a fim de gerar modelos que possam aprender a partir desses dados (MANIKAVELAN; PONNUSAMY, 2014; SHIVHARE; RATH, 2014).

Os modelos de EES também podem ser classificados conforme suas técnicas, sendo divididas em três (SHEPPERD, 2007):

1. centradas em humanos (ex. julgamento de especialistas);
2. baseadas em modelos, incluindo modelos algorítmicos/paramétricos (ex. COCOMO);
3. técnicas de predição induzidas.

Técnicas centradas em humanos, apesar de serem bastante utilizadas especialmente quando se tratam de modelos ágeis, apresentam problemas pois tornam-se bastante subjetivas, gerando muitas vezes estimativas distorcidas, com excesso de otimismo, por exemplo. A estimativa baseada em modelo é uma alternativa para as técnicas centradas em humanos, se apresentando como métodos reprodutíveis para gerar uma estimativa. Já as técnicas de predição induzidas, utilizam modelos de AM (ex. regressão linear, redes neurais, analogia) para formar a estrutura necessária para realizar a estimativa. Neste caso, os dados utilizados para treinamento dos modelos devem ser independentes dos dados utilizados em modelos algorítmicos/paramétricos (ex. COCOMO, Pontos por Função) (KOCAGUNELI *et al.*, 2011; MENZIES *et al.*, 2006; SHEPPERD, 2007).

A Estimativa de Esforço de Software por Analogia (EESA) tem sido amplamente utilizada e trata do processo de identificar um ou mais projetos históricos semelhantes ao projeto sendo desenvolvido e, em seguida, derivar a estimativa a partir deles. Este método foi proposto pela primeira vez em 1977 por Sternberg (1977) como uma alternativa válida para os pareceres de especialistas e estimativas de esforço algorítmicas (CHIU; HUANG, 2007).

2.1.1 Estimativa de Esforço de Software por Analogia (EESA)

A EESA, também conhecida por estimativa baseada em analogia, é um método não-paramétrico que utiliza dados de projetos passados para realizar uma estimativa para um novo projeto, sendo classificada por alguns autores (IDRI; AMAZAL; ABRAN, 2015) como uma técnica de AM, uma vez que ela vem sendo automatizada por meio do uso de modelos de AM. Além disso, os modelos AM, em especial aqueles que fazem uso de *Deep Learning* (DL) (por conta da não linearidade), também têm obtido uma atenção significativa de pesquisadores no campo da EES, pois com eles é possível mapear a relação complexa entre esforço e atributos de software (fatores de custo), especialmente quando essa relação não é linear e não parece ter qualquer forma predeterminada (IDRI; AMAZAL; ABRAN, 2015; KOCAGUNELI *et al.*, 2012).

Algumas vantagens de se utilizar modelos de EESA são (IDRI; AMAZAL; ABRAN, 2015):

- Tendem a produzir estimativas aceitáveis, superando outros modelos de estimativa, conforme indicado em estudos (IDRI; AMAZAL; ABRAN, 2015);
- Facilitam a modelagem de relações complexas entre o esforço e os atributos do software, podendo ser aplicada numa fase inicial de um projeto de software;
- Diferentemente de outras técnicas que utilizam Redes Neurais Artificiais (ANN), que são vistas como uma caixa-preta, as técnicas baseadas em analogia são indicadas por serem facilmente compreendidas pelos usuários, pois são semelhantes ao raciocínio humano por analogia.

A EESA pode ser utilizada tanto em modelos de desenvolvimento tradicionais (ex. Processo Unificado), como em modelos de desenvolvimento ágeis (ex. Scrum), uma vez que eles se baseiam em experiências anteriores da equipe para planejar novos projetos de software. O processo de estimativa por analogia, principalmente para modelos ágeis, normalmente é realizado por humanos, por meio de discussões entre os membros da equipe, não havendo uma utilização efetiva de dados históricos de projetos. Também por causa da característica ágil, muitas vezes esses dados não são registrados adequadamente (COHN, 2005).

Em especial, a metodologia ágil de software, amplamente utilizada por empresas de desenvolvimento de software nos mais variados portes (pequenas, médias ou grandes), é uma metodologia flexível que permite entregas mais rápidas aos *stakeholders* (pessoas interessadas no software), a partir de um processo mais simplificado, com geração reduzida de documentação, quando comparado a um processo de desenvolvimento de software tradicional. Nesse modelo de desenvolvimento, a mensuração de tamanho e a geração de estimativas de tempo e custos, na maioria das vezes, fica sob a responsabilidade da equipe envolvida no desenvolvimento, tornando-se bastante subjetiva (COHN, 2005).

2.2 Processamento de Linguagem Natural (PLN)

De acordo com Hirschberg e Manning (2015), a linguística computacional, também conhecida como Processamento de Linguagem Natural (PLN), é o subcampo da ciência da computação preocupado com o uso de técnicas computacionais para aprender, entender e produzir conteúdo de linguagem humana. De acordo com Russell e Norvig (2016), PLN consiste no desenvolvimento de modelos computacionais para a realização de tarefas que dependem de informações expressas em alguma língua natural (ex. tradução e interpretação de textos, busca de informações em documentos e interface homem-máquina).

Com a enorme quantidade de conteúdo de linguagem humana disponível atualmente e de forma online, algumas, dentre as muitas aplicações envolvendo PLN que vêm se destacando envolvem a análise de dados e extração de conhecimento (HIRSCHBERG; MANNING, 2015). Estas aplicações geralmente requerem a combinação de métodos de representação textual (ver Seção 2.4), com métodos de AM, de modo a converter o texto para um conjunto de características de AM, e utilizá-lo como base para a realização de tarefas de AM, respectivamente.

Para extrair e manipular adequadamente os significados de um texto, um sistema de PLN deve ter acesso a uma quantidade significativa de conhecimento obtido a partir de uma base de textos volumosa sobre o domínio do discurso, ou seja, sobre o contexto específico da área de aplicação que se busca trabalhar (CAMBRIA; WHITE, 2014). Esse conjunto de textos, utilizado na etapa de treinamento de sistemas de PLN pode ser chamado de corpus.

Um corpus é uma coleção de textos em um certo idioma, disponíveis em formato eletrônico (WYNNE, 2005; O'KEEFFE; MCCARTHY, 2010). Sendo assim, aplicações específicas

de PLN necessitam de uma base de dados textuais com uma grande quantidade de textos de referência no idioma e no domínio da aplicação em questão.

Ademais, uma grande limitação da PLN atualmente (MAGUERESSE; CARLES; HEETDERKS, 2020; SRIVASTAVA *et al.*, 2022), é o fato de que a maioria dos recursos e sistemas de PLN estão disponíveis apenas para idiomas de alto recurso, ou seja, que possuem uma grande quantidade de dados textuais classificados e preparados para o treinamento de modelos de AM, tais como (HIRSCHBERG; MANNING, 2015): inglês, francês, espanhol, alemão e chinês.

Em mineração de textos, os métodos para analisar e preparar textos em linguagem humana, são usados na etapa de pré-processamento de forma a melhor representar os recursos textuais e aproveitar mais o conteúdo. A seguir, são descritas as principais técnicas de pré-processamento de textos aplicadas em PLN.

2.3 Pré-processamento de Textos

A etapa de pré-processamento textual pode ser definida como uma sequência de decisões e ações sobre como as palavras deverão ser convertidas em números, visando tornar as entradas para uma determinada análise menos complexa, de uma maneira que não afete negativamente a interpretabilidade ou as conclusões substantivas do modelo subsequente (DENNY; SPIRLING, 2018).

De forma geral, as etapas do pré-processamento de texto envolvem: conversão dos documentos para a forma de texto simples sem formatação, ou seja, padronização do texto para letras minúsculas e remoção de acentuações e caracteres especiais, tokenização, remoção de *stopwords* e normalização de palavras (CONRADO *et al.*, 2014).

Já Kao e Poteet (2007) comentam que o PLN inclui técnicas de pré-processamento, como a remoção de palavras indesejadas (*stopwords*), *stemming* (remoção de sufixos, reduzindo uma palavra ao seu formato raiz), lematização (substituindo uma palavra flexionada por sua forma base), normalização de sinônimo, etiquetamento de partes da fala de acordo com as classes gramaticais (do inglês *Part-of-Speech* [POS]), entre outros.

Desta forma, verificando as etapas de pré-processamento citadas por ambos os autores (KAO; POTEET, 2007; CONRADO *et al.*, 2014), a seguir são descritas cada uma delas:

- Tokenização: consiste no processo de quebrar um fluxo de texto em palavras, frases, símbolos ou outros elementos significativos chamados *tokens*, visando principalmente a exploração das palavras em uma frase (KANNAN *et al.*, 2014).

- Remoção de *Stopwords*: são palavras consideradas irrelevantes nos textos – como ‘e’, ‘esse’, etc – e portanto sem importância na classificação de documentos (KANNAN *et al.*, 2014). Preposições, artigos e conjunções são exemplos de palavras que geralmente não possuem muita importância na classificação semântica do texto. A remoção de *stopwords* aumenta a concentração de termos específicos e relevantes à aplica-

ção, reduz o tempo de processamento e torna o corpus mais compacto (SCHOFIELD; MAGNUSSON; MIMNO, 2017).

- Normalização: a etapa de normalização de palavras consiste na remoção de sufixos e/ou prefixos de flexão – elementos linguísticos que visam expressar categorias gramaticais como modo, tempo ou gênero, por exemplo (CRYSTAL, 2011) – das palavras do texto, eliminando suas variações. Em aplicações de PLN, duas técnicas frequentemente utilizadas para a normalização são (TOMAN; TESAR; JEZEK, 2006):

    1. *stemming*: também chamada de radicalização, objetiva a redução das palavras de um texto para o seu radical (ex. estudamos $\rightarrow$ estud).
    2. lematização: consiste na redução de cada palavra de um texto para a sua forma canônica, ou seja, sem flexões (MANNING; RAGHAVAN; SCHÜTZE, 2008). Nessa etapa, os verbos são reduzidos ao infinitivo (ex. estudamos $\rightarrow$ estudar) e os substantivos e adjetivos reduzidos para o masculino singular (ex. alunas $\rightarrow$ aluno);

2.4 Representação Textual

Para que o computador consiga realizar a classificação de textos, extraindo conhecimento e contexto dos mesmos, estes devem ser convertidos em um formato que o computador possa entender (NASEEM *et al.*, 2021). Para tal, se aplicam modelos de representação textual, que objetivam representar uma palavra em um formato vetorizado (LIU; LIN; SUN, 2020).

Diversas formas de representação são encontradas na literatura, dentre os quais se destacam os modelos clássicos, que contemplam modelos categóricos e ponderados de representação, e os modelos de *word embeddings*, os quais são descritos a seguir nas Seções 2.4.1 e 2.4.2 (NASEEM *et al.*, 2021).

2.4.1 Modelos clássicos de representação de palavras

A representação categórica é a maneira mais simples de se representar um texto, na qual as palavras são transcritas para uma representação simbólica de "1"ou "0". Ainda, a codificação *One-hot* e o *Bag-of-words* (BOW), ou saco de palavras na tradução literal do Inglês, são os dois principais modelos que utilizam métodos de representação categórica. Ambos os modelos são descritos à seguir (NASEEM *et al.*, 2021).

O modelo BOW é uma representação que transcreve sentenças de um texto para vetores de tamanho fixo, por meio da contagem de quantas vezes cada palavra aparece nas mesmas. Ainda, cada índice do vetor está associado à contagem de quantas vezes certa palavra apare-

ceu, de tal forma que a informação da posição da palavra no texto é simplesmente descartada (GOLDBERG, 2017; NASEEM *et al.*, 2021).

De acordo com Goldberg (2017) e Naseem *et al.* (2021), o modelo de codificação *One-hot* pode ser considerado como uma instância atômica do modelo BOW – ou como um "saco de uma palavra só", como cita Goldberg (2017) –, uma vez que ele basicamente traduz uma palavra única em um vetor que recebe "1" no índice desta palavra, e "0" em todas as posições destinadas às outras palavras, funcionando de forma semelhante a um identificador. Assim, segundo os autores, cada dimensão do vetor corresponde a uma única característica, ou seja, uma única palavra.

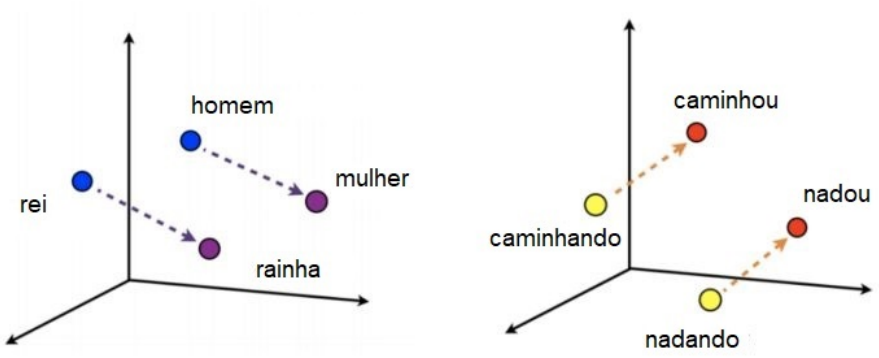
De acordo com Naseem *et al.* (2021), os modelos ponderados de representação textual podem ser divididos em Frequência do Termo, do Inglês *Term Frequency* (TF) e *Term Frequency-Inverse Document Frequency* (TF-IDF). Ainda segundo os autores, estes modelos são derivados dos métodos de representação categóricos como o BOW, tendo como principal diferença que, em vez de apenas contar, os modelos ponderados calculam um *score* (pontuação, em tradução literal do inglês) para cada palavra para avaliar sua importância no documento, baseado em sua frequência. Ambos os modelos são brevemente descritos a seguir.

Naseem *et al.* (2021) descrevem, de forma breve, que o método de representação textual TF mede a frequência das palavras em um documento, e realiza a sua normalização dividindo-a pelo número total de palavras neste documento, resultando em scores de [0, 1] (0 e 1 inclusive). Segundo os autores, em seguida os documentos são vetorizados considerando a lista de todas as palavras possíveis no corpus – e não apenas no próprio documento em si, o que resultaria em vetores de diferentes comprimentos –, de forma que cada posição do vetor originado corresponda a uma palavra, assim como o é no modelo BOW.

A representação TF não é tão eficiente quando há a presença de *stopwords* (ex. "e", "ou" e "o"), uma vez que estas palavras tendem a aparecer muitas vezes nos documentos do corpus, recebendo scores muitas vezes mais altos que das palavras realmente relevantes para a aplicação. Para amenizar este problema, a representação TF-IDF multiplica um termo logarítmico ao *score* TF das palavras, fazendo com que palavras que apareçam em vários documentos (ex. as *stopwords*) possuam um peso menor e palavras que aparecem muitas vezes em um número pequeno de documentos possuam um peso maior (MANNING; RAGHAVAN; SCHÜTZE, 2008; NASEEM *et al.*, 2021).

As representações BOW e *One-hot* são melhores utilizadas em casos em que há relativamente poucas palavras distintas na categoria, e acredita-se que não há correlação entre as diferentes palavras, uma vez que estas representações não trabalham tão bem com características correlacionadas e não levam em conta o contexto em que as palavras estão inseridas (GOLDBERG, 2017). Já as representações TF e TF-IDF, construídas sobre o conceito do modelo BOW, são melhores utilizadas para classificação de palavras no nível léxico, uma vez que também não pode capturar a ordem das palavras em um documento, semântica e informações sintáticas das palavras (NASEEM *et al.*, 2021).

Figura 1 – Exemplos de plot 3D de word vectors.



Fonte: Adaptado de Fonseca (2021)

Desta forma, alternativas mais modernas de PLN para a representação textual, e que oferecem um nível semântico do texto, são os modelos de *word embeddings* (FÁVERO; CASANOVA; PIMENTEL, 2022), os quais são apresentados a seguir.

2.4.2 *Word embeddings*

Um *word embedding* (ou apenas *embedding*) é uma representação aprendida para um texto, ou seja, que utiliza de métodos de redes neurais para a extração de características textuais (BENGIO; COURVILLE; VINCENT, 2013), a qual mapeia as palavras de tal forma que palavras que estão mais próximas no espaço vetorial sejam semelhantes em significado (JURAFSKY; MARTIN, 2000). Na área de PLN, métodos de representação de texto não supervisionados, como os *embeddings*, substituíram os métodos clássicos de representação de texto, justamente devido a esta sua capacidade de descobrir representações textuais de forma autônoma (NASEEM *et al.*, 2021).

Em contraste com os milhares ou milhões de dimensões necessárias para representações de palavras esparsas, geradas pelos modelos BOW, na representação por *embeddings*, cada palavra geralmente é representada por um vetor de valores reais com dimensões na ordem de dezenas ou centenas (GOLDBERG, 2017). A Figura 1 apresenta um exemplo de representação vetorial de *embeddings* para formas comparativas e superlativas de adjetivos e tempos verbais.

A partir da representação geométrica entre os *embeddings*, é possível capturar relações semânticas (por ex. entre um verbo no infinitivo e uma de suas flexões) (MIKOLOV *et al.*, 2013). Desta forma, torna-se possível representar a importância semântica de uma palavra no formato numérico (ZHANG *et al.*, 2015).

Pelo fato dos modelos de *word embeddings* representarem palavras por vetores com valores reais de forma que é possível atribuir uma palavra a um ponto em um espaço N-dimensional, similarmente ao que é ilustrado na Figura 1, torna-se possível realizar um cál-

culo de distância entre dois pontos para calcular a distância entre duas palavras (GOLDBERG, 2017), ou ainda, sua similaridade.

Ainda, dentre os métodos preditivos para geração de *embeddings* a partir de textos, é possível classificá-los em (HAJ-YAHIA; SIEG; DELERIS, 2019):

- sem-contexto (ou estáticos);
- contextualizados (ou dinâmicos).

A seguir, são detalhados de forma resumida os *word embeddings* sem-contexto e, em seguida, os *word embeddings* contextualizados.

2.4.2.1 *Word embeddings* sem contexto

De acordo com Barua *et al.* (2021), os modelos de *word embeddings* sem-contexto produzem apenas um vetor, prestando pouca auto-atenção à posição das palavras em uma sentença e às suas implicações. Os autores ainda complementam que, por exemplo, na frase “Ela andou pela margem do rio e foi até o banco para depositar dinheiro”, na qual a palavra ‘banco’ é utilizada de duas formas diferentes, esses modelos simplesmente juntariam ambas as conotações em um único vetor para ’banco’ durante sua execução.

O modelo de *embedding* não-contextualizado *Word to vector* (Word2Vec) implementa internamente duas abordagem utilizando redes neurais artificiais, o BOW Contínuo, do inglês *Continuous Bag of Words* (CBOW), que no seu treinamento analisa o contexto de cada palavra em uma sequência de texto e prevê uma palavra correspondente às palavras circundantes, e o SkipGram, que faz o inverso, ou seja, durante o treinamento tenta prever as palavras de contexto (ou palavras circundantes) para uma determinada palavra de entrada (BARUA *et al.*, 2021).

Já o método GloVe (*Global Vectors*) é fundamentado na ideia de que é possível derivar relações semânticas entre palavras por meio de uma matriz de co-ocorrência, que contém as frequências em que cada par de palavras aparecem em conjunto. Dado um corpus com N palavras, a matriz de co-ocorrência X é uma matriz NxN, na qual a i-ésima linha e a j-ésima coluna de X, X_{ij}, denota quantas vezes a palavra i co-ocorreu com a palavra j (PENNINGTON; SOCHER; MANNING, 2014).

Um exemplo de matriz de co-ocorrência é apresentado na Figura 2. Nela, é possível observar que nos índices atrelados a palavras relacionadas, como "gato"e "o", o valor 1 denota que existe relação, enquanto entre palavras que não possuem relação, como "gato"e "no", o valor no índice correspondente é 0.

A vantagem do modelo GloVe é que, ao contrário do Word2Vec, ele não depende apenas de estatísticas locais (ou seja, informação de contexto local das palavras), incorporando estatísticas globais (de co-ocorrência de palavras) para obter vetores de palavras (PENNINGTON; SOCHER; MANNING, 2014).

Figura 2 – Matriz de co-ocorrência para a sentença "o gato sentou no tapete".

	o	gato	sentou	no	tapete
o	0	1	0	1	1
gato	1	0	1	0	0
sentou	0	1	0	1	0
no	1	0	1	0	0
tapete	1	0	0	0	0

Fonte: Adaptado de Ganegedara (2021)

Ainda assim, estes modelos não fornecem um representação contextual completa para cada palavra no texto. Para tal, devem ser utilizados *embeddings* contextualizados, a qual são descritos a seguir.

2.4.2.2 *Word embeddings* contextualizados

Como já apresentado anteriormente, técnicas tradicionais de *word embeddings* constroem um vocabulário (ou dicionário), na qual seus elementos são palavras e seus *embeddings* correspondentes (BARUA *et al.*, 2021). Ainda, dada uma palavra, seu *embedding* é sempre o mesmo (sendo considerados estáticos), independente da sentença em que ela esteja inserida (JURAFSKY; MARTIN, 2019). Isso ocorre porque esses modelos são unidirecionais, ou seja, consideram o contexto de uma palavra apenas a partir da esquerda para a direita, sem nenhum mecanismo que detecte se uma determinada palavra já ocorreu antes ou não.

Além disso os modelos de *embeddings* tradicionais, segundo seus autores, são considerados muito rasos, pois representam cada palavra por apenas uma camada, havendo um limite para a quantidade de informações que eles podem capturar (MIKOLOV *et al.*, 2013).

Métodos de *embeddings* contextualizados, por outro lado, são utilizados para aprender semânticas no nível da sequência (ou sentença), pois consideram a sequência de todas as palavras em um documento. Essa abordagem produz diferentes representações vetoriais para a mesma palavra em um texto, o que varia de acordo com o seu contexto e, portanto, são capazes de capturar semânticas contextuais de palavras ambíguas, além de tratar as questões de polissemia[1] (AKBIK; BERGMANN; VOLLGRAF, 2019; JURAFSKY; MARTIN, 2019).

2.4.3 *Transfer Learning*

O *transfer learning* (ou transferência de aprendizado em tradução do Inglês) é uma técnica na qual utiliza-se um modelo de DL treinado em um conjunto de dados, geralmente extremamente volumoso (ex. Wikipédia), aproveitando o conhecimento deste modelo para outras

[1] fenômeno linguístico na qual uma palavra apresenta vários significados diferentes, dependendo do contexto em que a palavra está inserida (RIBEIRO, 2020)

tarefas. O resultado desse treinamento é um modelo pré-treinado, o qual adquiriu conhecimento necessário para executar tarefas semelhantes de diferentes conjuntos de dados ou mesmo de áreas distintas. Na área de Visão Computacional, modelos pré-treinados em conjuntos de dados de dados bastante volumosos (ex. Imagenet) já existem há mais tempo do que na área de PLN, área que apenas recentemente. Tanto como na área de Visão Computacional como em PLN, os modelos pré-treinados já foram aplicados em diversas tarefas, na qual obtiveram ótimos resultados (HOWARD; RUDER, 2018; ZHUANG *et al.*, 2021).

A indisponibilidade de grandes conjuntos de dados textuais rotulados para a realização de tarefas de PLN baseadas em aprendizado, foi um dos grandes motivos para que, até 2018, o PLN ficasse em atraso em relação à visão computacional (HOWARD; RUDER, 2018). Para mudar esse cenário, há alguns anos foram desenvolvidos os modelos pré-treinados com base em *Transformers* (auto-atenção) (VASWANI *et al.*, 2017): *Bidirectional Encoder Representations from Transformers* (BERT) (DEVLIN *et al.*, 2018) e *Transformer* Pré-treinado Generativo, do inglês *Generative Pre-trained Transformer* (GPT) (RADFORD *et al.*, 2018).

O modelo BERT foi treinado sobre uma enorme quantidade de dados de texto não rotulados. Sendo assim, esse modelo pode ser aplicado para realizar outras tarefas de PLN (ex. classificação de textos, reconhecimento de entidade nomeada (NER), geração de texto, entre outros). Desta forma, o *transfer learning* surgiu para revolucionar e definir uma nova era na área de PLN. O *transfer learning* demonstrou (HOWARD; RUDER, 2018) atingir um desempenho semelhante em comparação com um modelo não pré-treinado com até 10 vezes menos amostras de dados (DEVLIN *et al.*, 2018).

Já o modelo de linguagem contextualizado GPT e, consequentemente, seu sucessor GPT-3, também é baseado em *Transformers*, de modo que analisa a relação entre uma palavra e seus vizinhos em cada sentença. Assim, nestes modelos, as palavras não possuem *embeddings* estáticos, mas possuem *embeddings* gerados dinamicamente, além de aceitar frases inteiras como entrada, e não apenas palavras isoladas como os modelos sem-contexto (JURAFSKY; MARTIN, 2019).

Dada sua importância para a geração de *embeddings* contextualizados, a seguir são abordados os modelos BERT e GPT, dois dos principais modelos de representação pré-treinados contextualizados que têm sido utilizados pela comunidade de PLN.

2.4.4 Modelos de Representação Pré-treinados

O pré-treinamento de modelos de linguagem provou ser altamente eficaz no aprendizado de representações universais de linguagem a partir de dados não rotulados em grande escala (SUN *et al.*, 2019).

Os modelos de *embedding* pré-treinados são bem genéricos em sua representação e funcionam como uma base robusta de conhecimento para aplicações com pouca quantidade de dados ou dados esparsos (JURAFSKY; MARTIN, 2019; GARG; SHARMA; LIANG, 2020). Estu-

dos recentes (RADFORD *et al.*, 2018; RADFORD *et al.*, 2019; BROWN *et al.*, 2020) demonstram que seu uso possui a capacidade de aumentar a performance de modelos de PLN, melhorar a qualidade das representações, e ainda, permitir uma rápida especialização para uma tarefa.

Para construir um modelo pré-treinado, são utilizados grandes conjuntos de dados de linguagem natural de diversas áreas, que são pré-processados e preparados para serem dados como entrada no aprendizado do modelo. Após a etapa de pré-treino, esses modelos podem ser utilizados para transferir conhecimento para outras tarefas (JURAFSKY; MARTIN, 2019).

Dentre os modelos de *embeddings* pré-treinados mais utilizados atualmente, destacam-se o BERT (DEVLIN *et al.*, 2018) e a família GPT (RADFORD *et al.*, 2018). O motivo principal é que ambos se utilizam do mecanismo de *Transformers* (VASWANI *et al.*, 2017) (ou transformadores, em português).

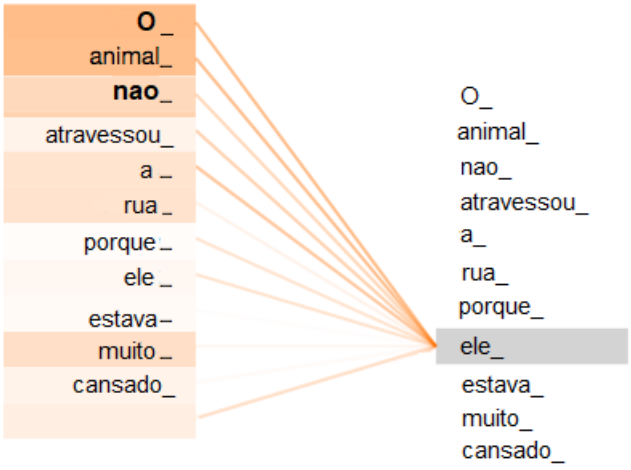
Os *Transformers* são um tipo de arquitetura de rede neural, que reduzem a necessidade da computação sequencial de sentenças textuais a partir de seus vários módulos de auto-atenção, que permitem a identificação mais precisa do contexto específico de cada palavra, com base nas palavras que compõe o texto. Ele pondera diferencialmente o significado de cada parte dos dados de entrada e processa toda a entrada de uma só vez. Sua arquitetura interna é composta por dois mecanismos, um codificador e um decodificador (respectivamente *encoder* e *decoder*, do inglês), ambos descritos com mais detalhes nas Seções 2.4.4.1 e 2.4.4.2, respectivamente (VASWANI *et al.*, 2017).

Na Figura 3 pode-se observar um exemplo de uso do mecanismo de auto-atenção, no processamento da frase: "o animal não atravessou a rua porque ele estava muito cansado". Nesta frase, a palavra “ele” (que na sua forma em Inglês, “*it*”, utilizada na figura original (ALAMMAR, 2018), não possui gênero) pode se referir tanto à palavra “rua” ou à palavra “animal”, o que computacionalmente caracteriza-se como uma indecisão. Deste modo, o mecanismo de auto-atenção atuaria processando a palavra "ele", buscando em seguida associá-la à palavra "animal", na mesma frase. Ainda, na Figura 3, palavras do lado esquerdo sinalizadas com tons de alaranjado mais intensos possuem uma maior probabilidade de relação com a palavra “ele” (destacada à direita), de acordo com a classificação do mecanismo de auto-atenção.

2.4.4.1 BERT

O *Bidirectional Encoder Representations from Transformers* (BERT) é um modelo de linguagem pré-treinado publicado pelos pesquisadores Devlin *et al.* (2018), do departamento de *AI Language* da Google. Segundo os autores, a inovação técnica do BERT é aplicar o treinamento bidirecional do *Transformer* (VASWANI *et al.*, 2017), um modelo de auto-atenção (do inglês, *self-attention*) popular aos modelos de linguagem pré-treinados contextualizados. Os resultados do artigo (DEVLIN *et al.*, 2018) mostram que um modelo de linguagem treinado bidirecionalmente pode ter um senso mais profundo de contexto e fluxo de linguagem do que modelos de linguagem unidirecionais.

Figura 3 – Exemplo de uso do mecanismo de auto-atenção.



Fonte: Adaptado de Alammar (2018)

Deste modo, o mecanismo *Transformer* permite analisar o contexto de cada palavra em um texto de forma individual, o que permite verificar se cada palavra já foi utilizada anteriormente em um mesmo contexto. Com isso, o método permite aprender relações contextuais entre palavras (ou sub-palavras) em um texto. Assim, o BERT é um modelo de linguagem em larga escala que consiste em várias camadas de blocos do tipo *Transformer*.

Isso se explica, porque o BERT tem se mostrado eficiente em diversas tarefas de PLN, pois utiliza o mecanismo codificador do *Transformer*, o qual, ao contrário dos modelos direcionais, que lêem a entrada de texto sequencialmente (da esquerda para a direita ou da direita para a esquerda), lêem toda a sequência de palavras de uma só vez e em ambas as direções, permitindo que o modelo aprenda o contexto de uma palavra com base em todos os seus arredores (esquerda e direita da palavra) (DEVLIN *et al.*, 2018).

Além disso, o BERT usa duas estratégias de treinamento: o *Masked Language Modeling* (MLM), na qual algumas das palavras em cada sequência são substituídas por um *token* de máscara e, em seguida, o modelo tenta prever o valor original delas com base no contexto fornecido pelas outras palavras (não mascaradas) na sequência, e o *Next Sentence Prediction* (NSP), na qual o modelo recebe pares de sentenças como entrada e aprende a prever se a segunda sentença do par é a sentença subsequente no documento original. Ao treinar o modelo BERT, os preditores do MLM e do NSP são treinados juntos, com o objetivo de minimizar a função de perda combinada das duas estratégias (DEVLIN *et al.*, 2018).

2.4.4.2 Família GPT

A arquitetura GPT (*Transformer* Pré-treinado Generativo, do inglês *Generative Pre-trained Transformer*), de acordo com seus autores (RADFORD *et al.*, 2018), também imple-

menta uma rede neural profunda (assim como o BERT), especificamente um modelo bidirecional de *Transformer*, que usa auto-atenção no lugar de arquiteturas anteriores baseadas em recorrência e convolução. Mecanismos de auto-atenção (VASWANI *et al.*, 2017) permitem que o modelo se concentre seletivamente em segmentos de texto de entrada que ele prevê serem os mais relevantes para um dado contexto (JURAFSKY; MARTIN, 2019).

A segunda versão do modelo de linguagem GPT da empresa OpenAI, GPT-2 (RADFORD *et al.*, 2019), foi lançada em novembro de 2019, e contava com 1,5 bilhão de parâmetros, constituindo um aumento de dez vezes na contagem de parâmetros, além de um aumento no tamanho do conjunto de dados de treinamento em relação ao seu antecessor, GPT-1, conseguindo uma considerável melhora no desempenho geral do modelo de linguagem em diversas tarefas (RADFORD *et al.*, 2019; SOLAIMAN *et al.*, 2019).

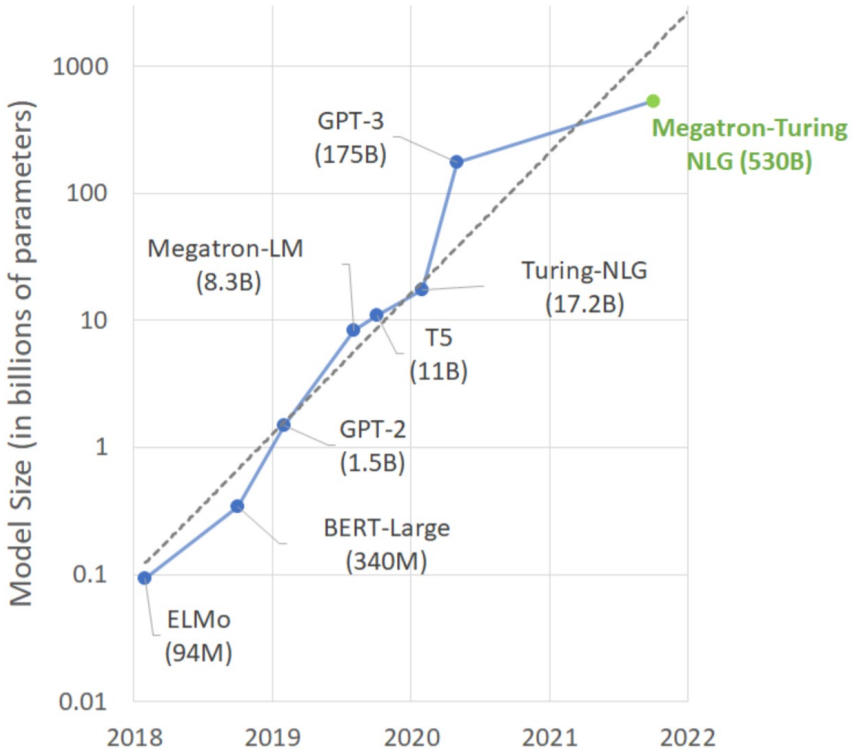
Já o GPT-3 (BROWN *et al.*, 2020), modelo mais recente da família GPT até o momento, conta com versões de 2,7 à 175 bilhões de parâmetros, um aumento de até mais que cem vezes comparado ao seu antecessor, além de conter textos de uma variedade maior de tópicos. Por conta disso, este modelo obteve uma melhor performance em configurações *zero-shot*, *one-shot* e *few-shot* em relação aos seus antecessores, ou seja, recebendo poucos ou nenhum exemplos da tarefa de destino, de tal forma que, em alguns casos, quase equipara-se ao desempenho de sistemas do estado da arte com ajuste-fino (BROWN *et al.*, 2020; RADFORD *et al.*, 2019)

Na Figura 4, é possível visualizar a tendência da evolução do número de parâmetros dentre os modelos de PLN mais recentes. Nela, é possível notar a grande diferença do modelo GPT-3 para o GPT-2 e o BERT-Large na quantidade de parâmetros.

Diferente do modelo BERT, que utiliza o codificador do *Transformer* original, o GPT e seus recentes sucessores GPT-2 e GPT-3 utilizam o decodificador do mesmo, sendo assim considerados modelos de linguagem auto-regressivos (DEVLIN *et al.*, 2018; RADFORD *et al.*, 2018). Neste tipo de modelo de linguagem, “a palavra gerada a cada passo de tempo é condicionada à palavra selecionada pela rede da etapa anterior”, ou seja, conforme as palavras de saída vão sendo geradas uma a uma pelo *Transformer*, elas vão sendo realimentadas como entrada para o mesmo, de forma a serem também consideradas na escolha da próxima palavra (JURAFSKY; MARTIN, 2019).

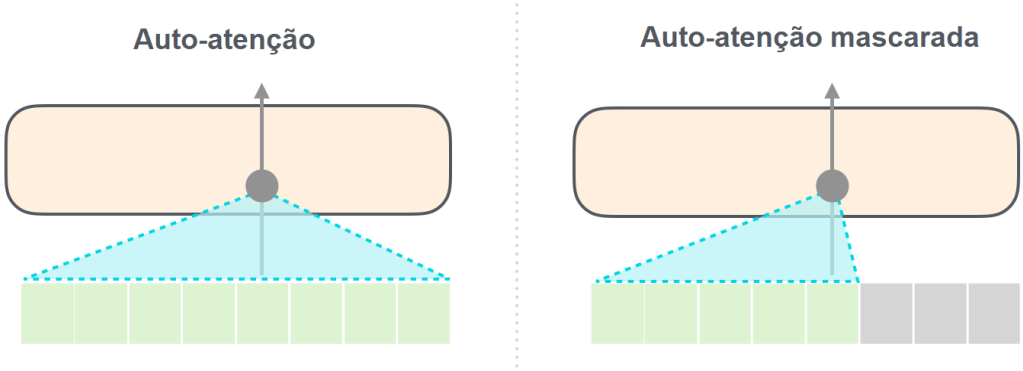
Outra fundamental diferença do modelo GPT para o BERT está na camada de auto-atenção, que mascara palavras futuras não substituindo palavras por máscaras diretamente, mas bloqueando informações de *tokens* que estão à direita da posição do *token* que está sendo processado (VASWANI *et al.*, 2017; RADFORD *et al.*, 2018). A Figura 5 faz a comparação entre a estratégia de auto-atenção dos dois modelos. No mecanismo de auto-atenção da esquerda, o modelo conhece os próximos *tokens* da sentença, ao contrário do mecanismo da direita, que é mascarado.

Figura 4 – Tendência de tamanhos de modelos de PLN de última geração com o tempo.



Fonte: Smith *et al.* (2022)

Figura 5 – Comparação entre auto-atenção e auto-atenção mascarada.



Fonte: Adaptado de Alammar (2019)

2.5 Algoritmos de aprendizado e métricas

De acordo com Haenlein e Kaplan (2019), a Inteligência Artificial (IA) pode ser definida como a capacidade de um sistema de interpretar dados de entrada, extrair conhecimento destes dados e, em seguida utilizar este conhecimento para a resolução de um problema ou tarefa específicos. Aplicações recentes se destinam às mais variadas áreas, e vão desde a robótica e veículos autônomos, até jogos, reconhecimento de voz e tradução automática (RUSSELL; NORVIG, 2016).

A IA pode ser vista como uma área mais abrangente e genérica, a qual engloba subáreas menores e específicas, dentre elas estão: o Aprendizado de Máquina (AM), que engloba as Redes Neurais Artificiais (ANN) e o *Deep Learning* (DL), e o Processamento de Linguagem Natural (PLN) (BENKE; BENKE, 2018).

2.5.1 Aprendizado de Máquina

O AM fornece aos sistemas a capacidade de aprender e melhorar automaticamente, com base na experiência, por meio da realização de treinamentos com dados relevantes à tarefa alvo. Para o processo de treinamento e predição, os métodos de AM costumam utilizar uma das seguintes abordagens: supervisionada, não supervisionada, de aprendizagem por reforço e semi-supervisionada. Independente de qual seja a abordagem, o objetivo encontra-se em gerar predições, classificações e/ou representações à partir de um conjunto de características de entrada (JORDAN; MITCHELL, 2015).

As aplicações dos algoritmos de AM são diversas. Dentre os métodos de AM, os supervisionados são utilizados para classificação e regressão, e os não supervisionados para agrupamento (do inglês, *clustering*), redução de dimensionalidade e associação. Além disso, o aprendizado dos algoritmos supervisionados é realizado por meio de dados rotulados, ou seja, que possuem exemplos reais de classificação para os algoritmos se basearem. Enquanto isso, os algoritmos não-supervisionados realizam seu treinamento com dados não rotulados, ou seja, sem o auxílio de exemplos de classificação reais, de modo que necessitam empregar alguma técnica para estimar seu desempenho de forma autônoma. Algoritmos semi-supervisionados são uma combinação de algoritmos supervisionados e não supervisionados, de forma a utilizarem tanto dados rotulados como não rotulados durante o treinamento. Já os algoritmos de aprendizagem por reforço funcionam de uma forma um pouco diferente, pois ao invés de aprender com rótulos, estes algoritmos recebem apenas uma indicação (ou *feedback*) do quão errada ou do quão correta uma ação está, geralmente na forma de um valor numérico, de forma semelhante ao que é realizado em sistemas de controle (JORDAN; MITCHELL, 2015).

Ainda, na abordagem supervisionada, os dados de treinamento assumem a forma de uma coleção de pares (x, y) e o objetivo é produzir uma previsão $y*$ (ex. “spam” ou “não spam”, probabilidade de câncer, etc.) em resposta a uma consulta $x*$ (ex. imagens, documentos, sequências de DNA, etc.) (JORDAN; MITCHELL, 2015).

Outra diferenciação importante na área de AM, diz respeito ao tipo da saída, ou característica, a ser estimada, sendo os problemas divididos em de classificação e de regressão. Apesar de ambos possuírem por objetivo geral mapear entradas para saídas, os problemas de classificação e regressão são diferentes. Enquanto o primeiro gera como valor de saída um rótulo de classe discreta (ex. spam ou não spam, maligno ou benigno, etc.), o segundo preocupa-se em mapear entradas para saídas reais valoradas contínuas (ex. o preço de um produto) (HASTIE; TIBSHIRANI; FRIEDMAN, 2009).

2.5.2 Redes Neurais Artificiais (ANNs)

As *Artificial Neural Networks* ANNs, ou Redes Neurais Artificiais em tradução literal do Inglês, propuseram um novo paradigma para o treinamento de modelos com aprendizado de máquina, dando origem à uma nova ramificação da IA com o mesmo nome. Desta forma, individualmente, cada ANN é um sistema computacional de aprendizado que utiliza uma sequência de funções, denominadas neurônios artificiais, para entender e mapear a relação (muitas vezes complexa) entre um conjunto de dados de entrada e uma saída desejada. O seu conceito foi inspirado pela biologia humana e a maneira como neurônios do cérebro humano funcionam de maneira conjunta para processar entradas sensoriais humanas (JAIN; MAO; MOHIUDDIN, 1996).

A primeira geração de ANN teve início com o *Perceptron* de Rosenblatt (1958) e era composta por apenas um único neurônio artificial. Por conta desta característica, o modelo possuía a limitação de não ser capaz de resolver problemas não linearmente separáveis (ex. a implementação da porta lógica XOR) (ROSENBLATT, 1958; MINSKY; PAPERT, 1969).

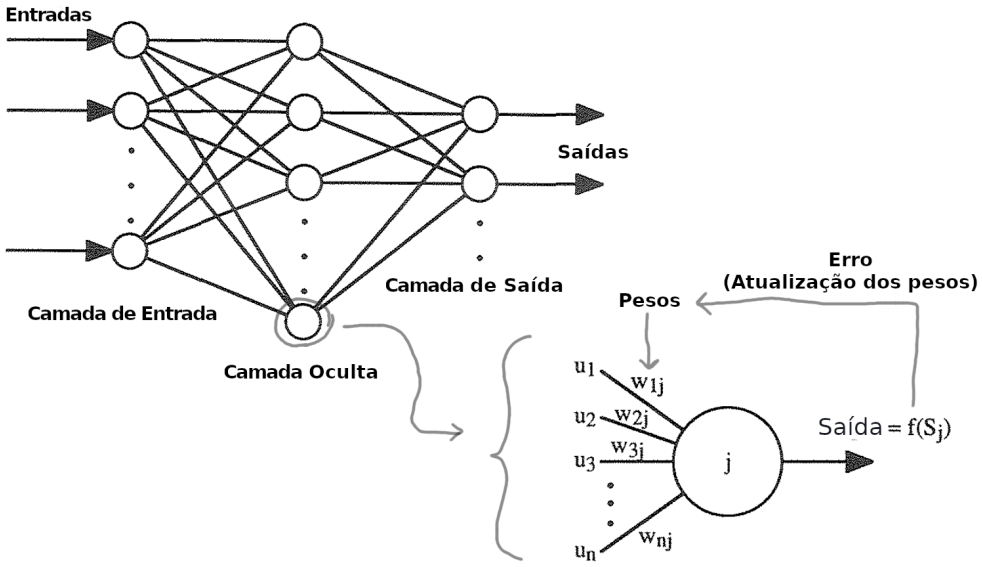
Deste modo, visando superar esta limitação, foram criadas as redes neurais de múltiplas camadas, também chamadas de Perceptron de Múltiplas Camadas, do Inglês *Multilayer Perceptron* (MLP) (RUMELHART; HINTON; WILLIAMS, 1986). As MLPs são uma rede neural do tipo *feedforward* [2], de modo que muitas vezes são chamadas simplesmente de Redes Neurais *Feedforward* (FNN). Portanto, as MLPs (ou FNNs, como serão referenciadas neste trabalho) são compostas por três componentes (JANCZAK, 2005):

- uma camada de entrada;
- uma ou mais camadas ocultas;
- e uma camada de saída.

A Figura 6 ilustra um exemplo de rede neural com uma camada oculta. Os neurônios da camada de entrada são responsáveis apenas pela recepção e distribuição dos dados de entrada pela rede, não realizando nenhum tipo de aprendizado sobre deles. Os neurônios das camadas ocultas recebem os valores de saída dos neurônios de sua camada anterior, valores estes que são multiplicados com um vetor de pesos, somados à um *bias* e aplicados à função de ativação deste neurônio, produzindo um valor de saída, conforme ilustrado pela Figura 6. Portanto, é dito que esses neurônios são responsáveis pelo aprendizado. Já os neurônios da camada de saída são responsáveis por analisar os dados obtidos das camadas ocultas e traduzi-los na inferência de um valor (ex. problemas de regressão) ou de uma classe (ex. problemas de classificação) (DAWSON; WILBY, 2009; GOODFELLOW; BENGIO; COURVILLE, 2016).

[2] Modelos de *Deep Learning feedforward* são assim chamados por não possuírem conexões de *feedback*, ou seja, na qual as saídas do modelo são realimentadas na entrada. Deste modo, a informação segue uma única direção dentro da rede neural, partindo da camada de entrada, passando pelas camadas ocultas e enfim para a saída do modelo. (GOODFELLOW; BENGIO; COURVILLE, 2016).

Figura 6 – Estrutura de uma FNN.



Fonte: Adaptado de Dawson e Wilby (2009).

Assim, a inferência se dá por meio da aplicação de múltiplas funções de ativação nas camadas ocultas, funções estas que são aprendidas iterativamente pelos neurônios da rede. Uma das principais maneiras de realizar o processo de aprendizado se dá da seguinte forma (GOODFELLOW; BENGIO; COURVILLE, 2016):

1. no começo do processo de treinamento da rede, os neurônios são inicializados com pesos aleatórios;

2. em seguida, são geradas estimativas para o conjunto de dados de treino, calculando-se o erro para cada par de valor estimado e valor esperado (ex. por meio do cálculo do gradiente do erro);

3. o valor do erro (também chamado de custo) é assim propagado pela rede, alterando o valor dos pesos dos neurônios, visando diminuir os valores obtidos pela função de custo (ex. função gradiente), conforme ilustrado na Figura 6.

2.5.2.1 Funções de Ativação

Funções de ativação, também chamadas de funções de transferência, são usadas em redes neurais para decidir se um neurônio deve ser ativado ou não. Essas funções são aplicadas após a soma ponderada de entrada e *bias*, e podem ser lineares ou não lineares. Por conta disso, são elas que introduzem a característica de não linearidade às representações aprendidas pelas redes neurais. Deste modo, sem funções de ativação, as redes neurais ficam limitadas à representação de funções lineares que, apesar de serem mais simples e fáceis de

se utilizar, possuem menor capacidade de representação e potencial de uso, uma vez que muitos problemas possuem dados com complexas relações não lineares (NWANKPA *et al.*, 2018; SZANDAŁA, 2021).

Atualmente, existem diversas funções de ativação, sendo algumas das mais conhecidas: linear, sigmoide, tangente hiperbólica (tanh), *softmax*, *softsign*, *swish*, Unidade Linear Retificada, do Inglês *Rectified Linear Unit* (ReLU) e *Exponential Linear Units* (*ELU*). Para problemas de regressão, utilizando-se uma função de ativação não linear nas camadas ocultas e uma função de ativação linear na saída, é possível melhorar a capacidade de representação do modelo de ANN, melhorando a precisão de sua estimativa. Ainda, estudos realizados com diferentes funções de ativação (BIRCANOĞLU; ARICA, 2018) para problemas de regressão e de classificação obtiveram melhores resultados em problemas de regressão com a combinação de funções ReLU e linear simples nas camadas ocultas e de saída, respectivamente (NWANKPA *et al.*, 2018; SZANDAŁA, 2021).

2.5.2.2 *Deep Learning*

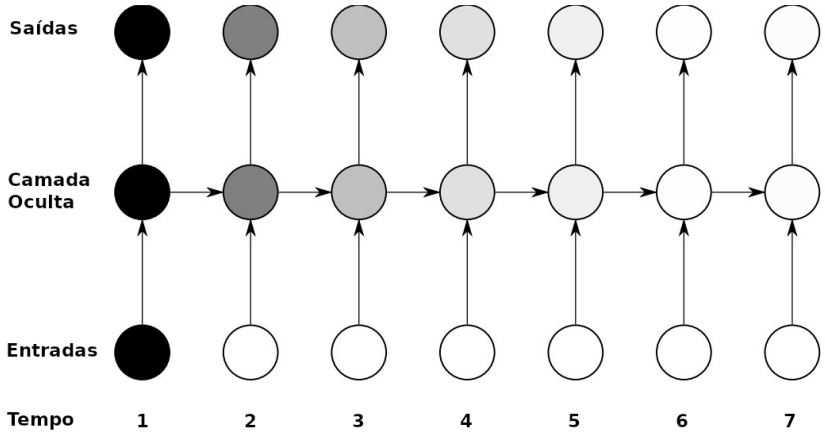
Nos últimos anos, o interesse da comunidade científica de aprendizado de máquina acerca de modelos de *Deep Learning* aumentou consideravelmente. A evolução das tecnologias computacionais é uma das responsáveis por sua popularização, uma vez que esta tornou possível a construção destes modelos, que apesar de relativamente antigos no campo teórico, eram computacionalmente complexos demais para serem implementados. Ademais, a variedade de aplicações em que as redes neurais profundas vêm se destacando também possui grande parcela de responsabilidade pelo aumento do interesse na área (HATCHER; YU, 2018).

De forma simplificada, o *Deep Learning* (DL) consiste em aprender de forma autônima a reconhecer a relação entre variáveis de entrada e de saída, e ainda, de forma autônoma adquirir os conhecimentos necessários para que esta relação seja válida. Ou seja, modelos de DL são especialistas aprender de forma autônoma uma função que mapeie dados de entrada numa saída desejada (ZHANG *et al.*, 2018).

2.5.2.3 Redes Neurais Recorrentes (RNNs)

Redes Neurais Recorrentes, do Inglês *Recurrent Neural Networks* (RNN) (RUMELHART; HINTON; WILLIAMS, 1986) são "redes neurais especializadas em processar dados sequenciais". Diferente das FNNs, as RNNs permitem conexões cíclicas em sua arquitetura. Desta forma, uma RNN consegue utilizar todo o histórico de entradas anteriores para gerar suas saídas, uma vez que as conexões de recorrência formam uma espécie de memória das entradas anteriores. Isso permite que as RNNs suportem sequências de entrada muito mais longas do que seria prático para redes FNN, por exemplo (GOODFELLOW; BENGIO; COURVILLE, 2016; GRAVES, 2012).

Figura 7 – Ilustração do problema do *vanishing gradient*



Fonte: Adaptado de Graves (2012).

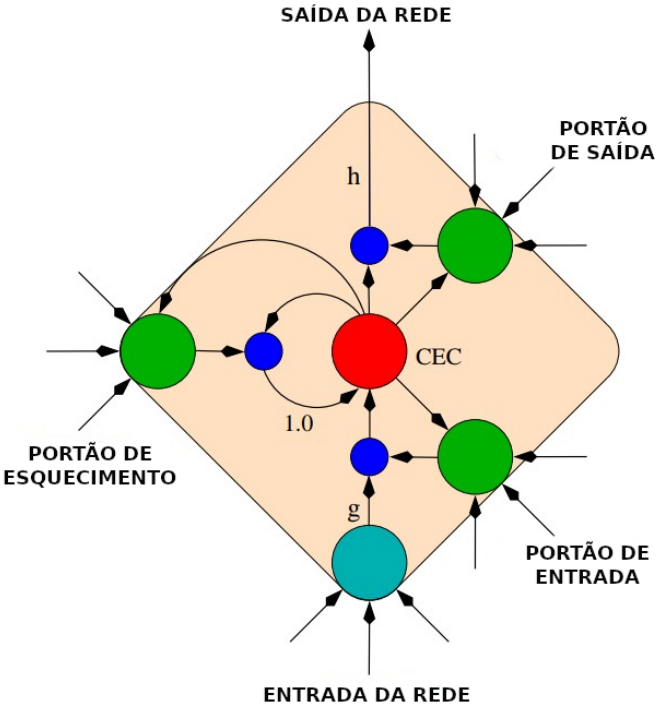
Apesar disso, uma grande limitação das redes RNN, é o problema do *vanishing gradient* (gradiente que desaparece, em tradução literal para o Português) (HOCHREITER, 1991; BENGIO; SIMARD; FRASCONI, 1994), que faz com que gradientes propagados por muitos estágios e, consequentemente, a influência de uma dada entrada na camada oculta e na saída da rede, tendam a decair ou crescer exponencialmente. Por conta deste problema, a janela com que o contexto de entradas anteriores pode influenciar entradas presentes em redes RNN é muito limitada em relação ao número de passos de tempo cobertos. Outra maneira de pensar sobre o problema, envolve imaginar a multiplicação de um peso w por ele mesmo t vezes, o que gera como resultado w^t. Como é de se imaginar, o produto w^t pode se tornar muito grande ou muito pequeno, dependendo da magnitude de w (HOCHREITER *et al.*, 2001; GOODFELLOW; BENGIO; COURVILLE, 2016; GRAVES, 2012).

O problema do *vanishing gradient* é ilustrado na Figura 7. Nela, a tonalidade dos nós indica a sensibilidade dos nós da rede (com o passar do tempo) à entrada do tempo um. Quanto mais escura a tonalidade, maior essa sensibilidade. Como pode-se observar na Figura 7, a sensibilidade diminui rapidamente conforme as novas entradas sobrescrevem a ativação da unidade oculta (GRAVES, 2012).

Para superar o problema do *vanishing gradient* foi criado o sistema de DL chamado de Memória Longa de Curto Prazo, do Inglês *Long Short-Term Memory* (LSTM) (HOCHREITER; SCHMIDHUBER, 1997). Diferente dos modelos de RNN tradicionais, os modelos LSTM realizam uma separação entre a saída de cada nó e a sua memória, e existem conexões autorecorrentes de *feedback* com um atraso temporal de um passo (HOUDT; MOSQUERA; NÁPOLES, 2020).

É importante ressaltar que existem algumas variações do modelo LSTM original, mas que a variação *vanilla* (GERS; SCHMIDHUBER; CUMMINS, 2000), é a mais popular dentre as aplicações que utilizam o LSTM, uma vez que ela demonstra boa performance em diferentes tarefas. Deste modo, a arquitetura LSTM *vanilla*, ou apenas "LSTM", como será chamada a partir de agora, é composta de uma célula com três unidades multiplicativas, denominadas portões

Figura 8 – Ilustração de um bloco de memória LSTM com uma célula



Fonte: Adaptado de Graves (2012).

(do Inglês, *gates*): entrada, esquecimento e saída, que fornecem operações análogas à escrita, reinicialização e leitura à célula de memória, respectivamente. Os portões utilizam uma função de ativação sigmoide, enquanto entrada e saída de cada bloco costumam utilizar uma função de ativação tangente hiperbólica (tanh) (GERS; SCHMIDHUBER; CUMMINS, 2000; GRAVES, 2012; HOUDT; MOSQUERA; NÁPOLES, 2020).

As redes LSTM de fato possuem uma arquitetura equivalente às RNN, exceto pelas unidades não-lineares nas camadas ocultas, que nas redes LSTM são substituídas por blocos de memória. Os portões multiplicativos das células de memória LSTM permitem que armazenem e acessem informações vindas de instantes de tempo maiores, de modo que o problema do *vanishing gradient* é evitado (GRAVES, 2012).

Na Figura 8 é possível visualizar a organização interna de um bloco de memória LSTM. O estado interno da célula, ou CEC (do Inglês, *Constant Error Carrousel*), é mantido por uma conexão recorrente com peso fixo de 1,0. Os portões de entrada e saída escalam (ou controlam) a entrada e saída da célula, enquanto o portão de esquecimento escala o estado interno da célula. Deste modo, enquanto o portão de entrada permanecer fechado, a ativação da célula não será sobrescrita por novas entradas na rede, podendo assim tornar-se disponível para a rede muito à frente na sequência, ao abrir o portão de saída (GRAVES, 2012).

2.5.2.4 Mecanismo de Atenção

Apesar de conseguirem trabalhar muito melhor com dependências entre entradas mais distantes temporalmente do que os modelos RNN tradicionais, as redes LSTM ainda assim possuem uma janela de tempo limitada na qual conseguem detectar essas dependências com maior precisão, fato testemunhado em estudos (BAHDANAU; CHO; BENGIO, 2014) envolvendo modelos para tradução de texto. No contexto de modelos de PLN, outro problema envolvendo tanto redes RNN tradicionais como redes LSTM é o fato de ambas precisarem comprimir todas as informações necessárias da sentença de entrada em um vetor de tamanho fixo, o que também compromete a performance destes modelos conforme as sentenças de entrada se tornam maiores (BAHDANAU; CHO; BENGIO, 2014).

Felizmente, uma solução para estes problema é a utilização do Mecanismo de Atenção (BAHDANAU; CHO; BENGIO, 2014; VASWANI *et al.*, 2017) (ver Subseção 2.4.4), que conseguem localizar as partes da sentença de entrada que são mais importantes e utilizá-las para realizar a predição (BAHDANAU; CHO; BENGIO, 2014).

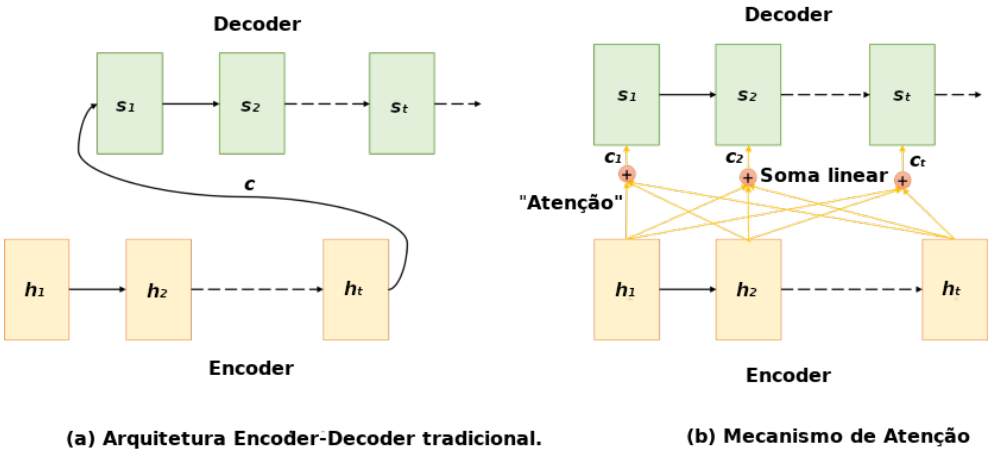
Para entender as melhorias trazidas pelo mecanismo de atenção, é necessário antes entender as diferenças de sua arquitetura para os *Encoder-Decoders* (ou Codificador-Decodificadores, em tradução literal do Inglês) RNN/LSTM. E para isso, também é necessário entender o que é um *Encoder-Decoder*.

De forma simplificada, a arquitetura *Encoder-Decoder*, também denominada *sequence to sequence* (sequência para sequência, em tradução literal do Inglês), recebe como entrada uma sequência de entrada, por meio de uma rede de *encoder*, a qual é convertida em uma representação contextualizada, que é comumente denominada de contexto. Em seguida, o vetor de contexto, c, é então passado para uma rede *decoder*, sendo assim convertido para uma sequência de saída *task-specific*, ou seja, específica de uma tarefa ou aplicação. Os modelos *Encoder-Decoder* podem ser implementados com RNNs ou com *Transformers* (JURAFSKY; MARTIN, 2019).

Ao comparar a arquitetura dos *Encoder-Decoders* RNN/LSTM para os Mecanismos de Atenção, a principal diferença notada é a adição de uma unidade *Softmax* (função de ativação não linear, comumente utilizada para classificação) no Mecanismo de Atenção, que recebe como entrada uma soma ponderada das saídas dos estados ocultos do *Encoder*. Esta unidade *Softmax* é responsável por fornecer o valor da utilidade de cada estado oculto do *Encoder* e portanto, a relevância de cada parte da sentença de entrada para o *Decoder* na geração do próximo *token* de saída (JURAFSKY; MARTIN, 2019).

Essas diferenças estão ilustradas na Figura 9, que compara (a) a arquitetura *Encoder-Decoder* tradicional com (b) o Mecanismo de Atenção.

Figura 9 – Diferença entre arquiteturas (a) *Encoder-Decoder* tradicional (RNN/LSTM) e (b) *Attention*.



Fonte: Adaptado de Yadav (2019).

2.5.3 Métricas de Avaliação

As métricas de avaliação são utilizadas para avaliar o desempenho de modelos de inferência, e são calculados com a distância entre os valores reais (do conjunto de teste) e os valores previstos, verificando assim, sua capacidade de generalização. A aplicação de métricas de avaliação é uma das etapas mais importantes do *pipeline* da construção de sistemas de AM, "uma vez que orienta a escolha do método ou modelo de aprendizagem e fornece uma medida da qualidade do modelo finalmente escolhido". Para a avaliação de modelos de previsão de esforço de software baseados em regressão, as métricas recomendadas são: Erro Absoluto Médio, do Inglês *Mean Absolute Error* (MAE), Erro Absoluto Mediano, do Inglês *Median Absolute Error* (MdAE) e Erro Quadrático Médio, do Inglês *Mean Squared Error* (MSE) (CHOETKIERTIKUL *et al.*, 2019; HASTIE; TIBSHIRANI; FRIEDMAN, 2009; RASCHKA, 2015; SARRO; PETROZZIELLO; HARMAN, 2016).

O MAE é uma medida da diferença entre duas variáveis contínuas, e é calculado realizando-se a média do módulo das diferenças entre os valores esperados e reais, sendo definido pela equação (CHOETKIERTIKUL *et al.*, 2019):

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

Neste e em ambas as métricas descritas a seguir, *N* é o número de exemplos de teste, y_i e $\hat{y}_i$ são os rótulos e a previsão da i-ésima amostra, respectivamente.

O MdAE é sugerido como uma métrica mais robusta para grandes valores discrepantes. Ela é calculada pela mediana do conjunto de diferenças absolutas entre os valores estimados e reais, sendo definida pela equação abaixo, na qual $1 \leq N \leq N$ (CHOETKIERTIKUL *et al.*, 2019).

$$MdAE = mediana\{|y_i - \hat{y}_i|\}$$

O MSE é uma medida quantitativa, útil para comparar diferentes modelos de regressão. Um valor de MSE igual a zero é o esperado, ou seja, quanto mais próximo de zero, melhor o desempenho do modelo (RASCHKA, 2015).

Além disso, o MSE é definido como a raiz quadrada da média das diferenças quadráticas entre os valores esperados e reais, conforme definido pela equação a seguir (RASCHKA; MIRJALILI; SAFARI, 2019):

$$MSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

É importante destacar a necessidade de realizar a média das métricas obtidas para o caso de utilização de métodos de validação cruzada, bem como o seu desvio padrão. Além disso, é importante escolher e calcular os Intervalos de Confiança para cada métrica individual, de modo a fornecer uma estimativa do grau de confiança de cada medida.

2.5.3.1 Intervalos de Confiança

Os Intervalos de Confiança (IC) introduzem um grau de incerteza a estimativas locais, de modo que possa-se especular o valor real da variável sendo analisada. Ao serem calculados para médias amostrais, os IC são construídos de maneira que a média populacional se encontra dentro de seus limites com uma certa probabilidade. Para construí-los, é necessário escolher a probabilidade com que se deseja que a média populacional se encontre dentro do intervalo. Geralmente probabilidades de 95% ou 99% são escolhidas (LANE, 2003).

Para um IC de 95%, utilizam-se as seguintes equações para o cálculo do limite inferior e superior, respectivamente (LANE, 2003):

$$L_{\text{inferior}} = M - Z_{0,95}\sigma_{\text{M}}$$

$$L_{\text{superior}} = M + Z_{0,95}\sigma_{\text{M}}$$

Onde M é a média amostral, $Z_{0,95}$ é o "número de desvios padrões que se estendem da média de uma distribuição normal requiridos para conter 95% da área", e σ_{M} é o desvio padrão da média M. Deste modo, os ICs de 95% podem ser interpretados como intervalos de confiança no qual, dado uma amostra completamente nova, existe uma probabilidade de 95% dela estar contida neste intervalo (LANE, 2003).

2.5.4 Métodos de Validação e Particionamento de Dados

Durante o treinamento dos modelos de AM, é possível dividir a base de dados em *datasets* (ou conjuntos de dados, na tradução literal do Inglês) de: treino, validação e teste. Com os *datasets* de treino e validação é realizado o ajuste de hiperparâmetros de configuração do modelo, visando aumentar sua precisão e capacidade de generalização. Existem diferentes métodos que podem ser utilizados nesta etapa, os quais dividem-se em analíticos (ex. Critério de Informação de Akaike, Critério de Informação Bayesiano, Descrição de Comprimento Mínimo e Minimização do Risco Estrutural) e por reutilização eficiente da amostra (ex. validação cruzada e *bootstrap*) (HASTIE; TIBSHIRANI; FRIEDMAN, 2009).

No caso dos métodos analíticos, uma estimativa do erro de predição do modelo final é obtida somando-se o erro obtido na fase de treinamento do modelo, com o cálculo de uma estimativa do otimismo deste erro em relação ao erro de generalização (ou seja, o erro obtido no processamento de novas amostras). Vale destacar que o erro obtido durante a fase de treinamento não é considerado relevante para a avaliação do desempenho de modelos, uma vez que amostras futuras provavelmente não irão coincidir com as mostras vistas durante esta fase (HASTIE; TIBSHIRANI; FRIEDMAN, 2009).

Já nas técnicas de validação cruzada e *bootstrap*, o erro para novas amostras é estimado diretamente durante a avaliação do modelo de inferência, de modo que estes métodos podem ser considerados mais eficientes na tarefa de avaliação do erro de generalização do modelo (HASTIE; TIBSHIRANI; FRIEDMAN, 2009).

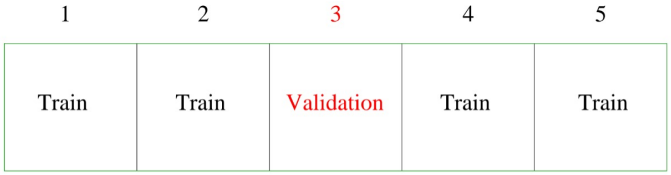
Ainda, de forma geral, estes métodos podem ser utilizados como meios para a avaliação do impacto da mudança de hiperparâmetros de configuração e, em alguns casos (ex. método de validação cruzada *k-fold*), podem estabelecer procedimentos para o particionamento dos dados. Em alguns casos, eles também são utilizados na etapa de teste, servindo como um método de avaliação da qualidade da estimativa do modelo final (HASTIE; TIBSHIRANI; FRIEDMAN, 2009; TRAPPENBERG, 2019).

O método de validação cruzada *k-fold* divide os dados de uma maneira que utiliza um dos subconjunto para avaliação, e todos os outros para realizar o treinamento do modelo, de modo que cada subconjunto é utilizado uma vez como subconjunto de validação/teste. Com isto, esse método procura resolver o frequente problema da escassez de dados para realização das etapas de validação e teste. Por outro lado, quanto maior o número de partições k, maior o número de iterações realizadas pelo método e, portanto, maior o tempo e os recursos necessários para a avaliação do modelo, de modo que o valor de k deve ser escolhido conscientemente (HASTIE; TIBSHIRANI; FRIEDMAN, 2009).

Deste modo, a Figura 10 ilustra o particionamento dos dados pelo método *k-fold* com um k igual a cinco (5). De forma geral, este método é aplicado da seguinte forma (WITTEN; FRANK; HALL, 2011):

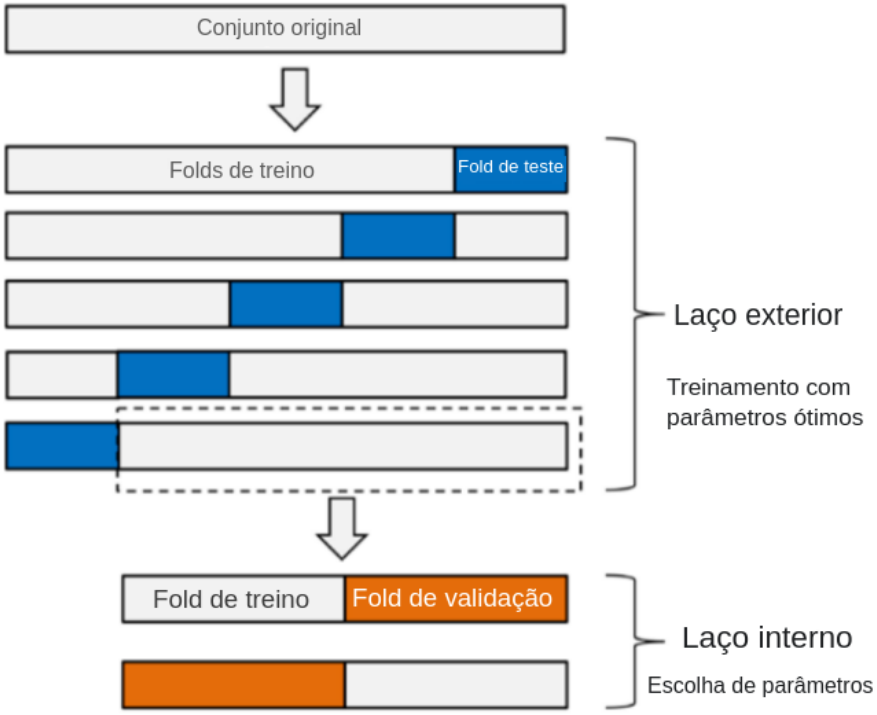
1. Os dados são divididos em k subconjuntos de igual tamanho;

Figura 10 – Exemplo de particionamento *k-fold* com um k igual a 5.



Fonte: Hastie, Tibshirani e Friedman (2009)

Figura 11 – O conceito de validação cruzada aninhada *k-fold*



Fonte: Adaptado de Raschka *et al.* (2022)

2. Cada um dos subconjuntos é usado uma vez para teste e os demais para treinamento;

3. É feita a média com os diferentes erros obtidos.

Mais completo que o método de validação cruzada *k-fold* tradicional, é o método de validação cruzada *k-fold* aninhada (do Inglês, *nested cross-validation*). Ele consiste na realização de dois laços de particionamentos *k-fold*, sendo o primeiro, externo, responsável por dividir os dados em *folds* (ou partições) de treino e teste, e o interno responsável pela seleção dos hiperparâmetros do modelo dentro do *fold* de treinamento, conforme ilustrado pela Figura 11. Desta forma, os métodos de validação cruzada aninhados são utilizados quando deseja-se empregar simultaneamente validação cruzada para seleção dos hiperparâmetros e estimativa do erro (AHAD; Das Antar; AHMED, 2020; RASCHKA *et al.*, 2022).

2.5.5 Otimização de Hiperparâmetros

Outra etapa a ser considerado no treinamento de modelos de inferência, é a etapa de Otimização de Hiperparâmetros, do Inglês *Hiper-Parameter Optimization* (HPO). Nesta etapa, procura-se encontrar a combinação de hiperparâmetros que apresente o menor erro de generalização. Realizar esta otimização de forma global, de modo a encontrar os melhores hiperparâmetros para um dado modelo de inferência é praticamente impossível, uma vez que é necessário conhecer todas as entradas possíveis para o mesmo. Deste modo, para tornar a HPO possível, são utilizados métodos de validação cruzada para estimar o erro de generalização, e utiliza-se uma amostra finita de dados (divididos em conjuntos de teste e validação) como entrada para o algoritmo (BERGSTRA; BENGIO, 2012).

Os algoritmos de HPO que constituem o estado da arte podem ser divididos em dois grupos: algoritmos de busca (ex. *grid search*, *random search*), utilizados para amostragem, e *trial schedulers* (ex. *HyperBand*, *Asynchronous Successive Halving*), que fazem uso de técnicas de *early stopping* [3]. (YU; ZHU, 2020).

O algoritmo de busca *grid search* é "um método básico de HPO", que realiza uma busca exaustiva no conjunto de hiperparâmetros de entrada. Apesar de ser um algoritmo popular por sua simplicidade e facilidade de uso, o *grid search* torna-se computacionalmente pesado à medida que o espaço de busca aumenta exponencialmente, uma vez que ele testa todas as combinações possíveis neste espaço. Assim, ele é melhor utilizado quando há não mais que três hiperparâmetros precisam ser ajustados simultaneamente, com poucos valores possíveis para cada hiperparâmetro (YU; ZHU, 2020).

2.6 Trabalhos Relacionados

A fim de entender e elucidar o estado da arte da EESA com base em textos, a pesquisa bibliográfica compreendeu o período de 2007 ao primeiro semestre de 2022. Destaca-se que esse mapeamento foi realizado no ano de 2018 (FÁVERO *et al.*, 2018) e foi complementado utilizando-se da mesma metodologia em 2022 (FÁVERO; CASANOVA; PIMENTEL, 2022) (conforme o Anexo A), incluindo o período de 2019 a 2022.

Sendo assim, a partir do mapeamento realizado, observa-se que a EESA é bastante promissora, pois é a que mais se assemelha ao método humano de fazer estimativas, ou seja, toma como base experiências anteriores e se utiliza de alguma técnica de analogia para obter o esforço mais aproximado. Outro dado importante e que revelou uma lacuna de pesquisa, é o baixo volume (13%) de utilização de requisitos textuais gerados nas fases iniciais do processo

[3] A estratégia de *early stopping* é utilizada para maximizar o uso de recursos computacionais para conjuntos de hiperparâmetros promissores. Ou seja, algoritmos que à utilizam permitem usuários a terminar uma avaliação antes de finalizar o treinamento completo, liberando assim recursos computacionais para avaliações com conjuntos de hiperparâmetros promissores (YU; ZHU, 2020).

de desenvolvimento para obtenção de características necessárias aos métodos de aprendizado de máquina. A grande maioria dos estudos analisados (87%) exploram artefatos textuais, utilizando texto de requisitos (ex. histórias de usuário) e sua estimativa de esforço, dada em alguma métrica padrão (ex. pontos por história), aliados ou não a atributos dessas histórias de usuário (FÁVERO *et al.*, 2018).

Ainda, observa-se que poucos estudos utilizam características de textos para a geração da EES mais precisas em modelos de estimativa baseados em analogia. Além disso, na maioria dos estudos analisados, são aplicadas abordagens de representação textual do tipo BOW, fazendo uso de características numéricas estáticas (ex. TF, TF-IDF). Alguns dos estudos também agregam aspectos sintáticos (ex. *Part-of-Speech* [POS]), não empregando conhecimento específico sobre a estrutura do texto dos requisitos e ignorando aspectos semânticos. O Quadro 1 mostra o resumo desses trabalhos, destacando: o tipo de análise textual, dados textuais utilizados e técnica de representação textual (FÁVERO; CASANOVA; PIMENTEL, 2022).

Analisando o Quadro 1, percebe-se a evolução das técnicas ao longo do tempo, partindo de abordagens estatísticas e sintáticas (ex. TF, TF-IDF) para, mais recentemente, abordagens semânticas, que fazem uso de *word embeddings* contextualizados (ex. Word2Vec, BERT). Entretanto, percebe-se que ainda há muito para ser estudado na área de processamento e geração de recursos textuais no campo da EESA (FÁVERO *et al.*, 2018).

De todos os estudos analisados, somente dois se destacam quanto à forma de representação das características textuais: Choetkiertikul *et al.* (2019) e Ionescu, Demian e Czibula (2017). Estes estudos aplicaram modelos de *word embeddings* para a representação textual, permitindo considerar aspectos semânticos de forma mais completa (FÁVERO; CASANOVA; PIMENTEL, 2022).

Ionescu, Demian e Czibula (2017) propuseram um método automatizado baseado em AM para EESA com base no texto dos requisitos. Para isso, seu *pipeline* responsável pelo aprendizado é composto por um método de codificação *one-hot* e cálculo do TF-IDF modificado (Doc2Vec) (FÁVERO; CASANOVA; PIMENTEL, 2022).

Enquanto isso, o trabalho de Choetkiertikul *et al.* (2019) propôs o uso de DL, pelo seu desempenho promissor na solução de problemas em diversas áreas. Desta forma, os autores combinaram dois modelos de ANN na arquitetura de DL: redes Memória Longa de Curto Prazo, do Inglês *Long Short-Term Memory* (LSTM) e *Recurrent Highway Networks* (RHN). Para o treinamento de seu modelo, os autores utilizaram dados de entrada brutos pré-processados. Deste modo, seu modelo utilizou as estimativas de pontos por história recuperados de projetos anteriores para adquirir conhecimento e tornar-se capaz de realizar previsões para novas histórias de usuário (FÁVERO; CASANOVA; PIMENTEL, 2022).

Além disso, Choetkiertikul *et al.* (2019) utilizaram *word embeddings* (não contextualizados) a nível de palavra como entrada na camada LSTM. Ademais, para a preparação dos dados de entrada para seu modelo, foram combinados o título e a descrição de um relatório de requisitos, nessa ordem, em uma única sentença. É importante ressaltar que a técnica de *word*

Quadro 1 – Utilização de texto para estimativa de esforço/tipo de análise e técnica de representação textual.

ID da publicação	Tipo de análise textual (Dados textuais utilizados)	Técnica(s) de representação textual	Modelo(s) de representação textual
Abrahamsson *et al.* (2011)	estatística	palavras-chave e respectivas frequências (TF)	BOW
Hussain, Kosseim e Ormandjieva (2013)	sintática	palavras-chave, TF e TF-IDF + extração de recursos linguísticos (ex. número de verbos, substantivos, sentenças, parênteses, entre outros)	BOW
Moharreri *et al.* (2016)	estatística	palavras-chave, TF e TF-IDF	BOW
Zhang *et al.* (2016)	semântica	análise semântica a nível de palavra (com *WordNet*)	BOW
Ochodek (2016)	sintática, semântica	POS, marcação de palavras, lematização e análise de dependências entre palavras (*Stanford Parser*)	BOW
Ayyildiz e Koçyigit (2016)	sintática	número de substantivos e verbos do domínio do problema e sua relação com o número de classes e métodos do domínio da solução	BOW
Ionescu, Demian e Czibula (2017)	estatística, semântica	*word embedding* sem contexto (*doc2vec*)	BOW + *word embedding*
Choetkiertikul *et al.* (2019)	semântica	*word embedding* sem contexto a nível de palavra + representação vetorial a nível de texto (via LSTM)	*word embedding*
Fávero, Casanova e Pimentel (2022)	semântica	*word embedding* com e sem contexto a nível de palavra + representação vetorial a nível de texto (via LSTM)	*word embedding*

Fonte: Adaptado de Fávero, Casanova e Pimentel (2022)

embedding utilizada pelos autores faz com que o modelo aprenda a gerar representações textuais de forma autônoma, com a própria experiencia, uma vez que a mesma é baseada em ANN e DL. Desse modo, as estimativas reais atribuídas a cada requisito textual alimentam o modelo, servindo como referência para o seu aprendizado (FÁVERO; CASANOVA; PIMENTEL, 2022).

Por fim, o trabalho de Fávero, Casanova e Pimentel (2022) apresenta o método SE^3M, o qual objetivou realizar uma comparação com os resultados obtidos por Choetkiertikul *et al.* (2019), fazendo uso da mesma base de dados. Fávero, Casanova e Pimentel (2022) aplicou o modelo pré-treinado BERT para realizar a representação textual, o qual foi dado como entrada em uma arquitetura DL baseada em LSTM.

Deste modo, nota-se que grande parte dos estudos se baseiam em recursos estatísticos e/ou sintáticos do texto, não empregando conhecimento específico sobre a estrutura do texto dos requisitos, ou seja, ignorando a ordem de ocorrência das palavras e outros aspectos semânticos ou de contexto. Apesar de Choetkiertikul *et al.* (2019) e Ionescu, Demian e Czibula (2017) considerarem aspectos semânticos pelo uso de *word embeddings*, ambos utilizaram modelos de *embeddings* sem-contexto. Fávero, Casanova e Pimentel (2022) apresenta uma melhoria ao modelo apresentado por Choetkiertikul *et al.* (2019) ao aplicar um modelo pré-treinado contextualizado (BERT). Sendo assim, a proposta desse trabalho é aplicar um modelo pré-treinado contextualizado melhorado, integrante do estado-da-arte na área de representação textual: o modelo GPT-3.

3 MATERIAIS E MÉTODO

Este capítulo lista os materiais e os métodos empregados no desenvolvimento deste trabalho, nessa ordem, além de justificar e descrever brevemente as principais ferramentas, tecnologias e métodos escolhidos.

3.1 Materiais

Os materiais, incluindo as ferramentas e tecnologias utilizadas para o desenvolvimento deste trabalho, encontram-se listados no Quadro 2.

A linguagem de programação Python foi escolhida para a implementação deste trabalho pois, além de possuir suporte à API (Interface de Programação de Aplicação, do Inglês *Application Programming Interface*) do GPT-3 da OpenAI, ela possui várias bibliotecas dedicadas a tarefas de PLN e AM, dentre elas a NLTK, o Pandas e o Scikit-learn.

Ainda, o Python é a linguagem utilizada pelo *Google Colaboratory* (ou Google Colab, como é mais conhecido), uma ferramenta da Google que permite escrever e executar códigos (em Python) em uma máquina virtual por meio do navegador, sendo muito utilizada para AM e análise de dados (incluindo também, PLN). O Google Colab foi utilizado pois, além das características já mencionadas, ele vem com a maioria, se não todas as bibliotecas utilizadas neste trabalho instaladas e configuradas por padrão, e também por possibilitar a utilização de forma gratuita. Ademais, suas máquinas virtuais permitiram executar os códigos das etapas de Coleta de Dados e Pré-processamento, e Extração de Características, descritas na Seção 3.2.

Já para o código de treinamento e avaliação do Modelo de Inferência (ver Subseção 3.2.4), procurando maximizar o número de parâmetros testados pelo método de validação cruzada, a execução foi realizada em um computador com as especificações listadas no Quadro 3.

A seguir são descritas brevemente as bibliotecas Python que foram utilizadas para a implementação do trabalho.

- Keras (CHOLLET *et al.*, 2015): uma biblioteca simples, mas poderosa, que se aproveita do TensorFlow, da Google, para criar modelos de aprendizado de máquina e de *Deep Learning* (GULLI; KAPOOR; PAL, 2019). Assim, ela foi utilizada para a configuração e o treinamento do modelo de redes neurais que posteriormente foi utilizado pela predição do esforço de software.

- Matplotlib (HUNTER, 2007): biblioteca, que permitiu a geração de histogramas à partir dos dados dos requisitos textuais e suas estimativas de esforço.

Quadro 2 – Tecnologias utilizadas para o trabalho

Ferramentas/Tecnologias	**Versão**	**Finalidade**
Google Colab	2021	Servidores da Google (máquinas virtuais), que permitem escrever e executar código em Python no navegador de forma rápida e simples
GPT	3	Algoritmo pré-treinado de aprendizado profundo (*Deep Learning*) da OpenAI para PLN.
Keras	1.2	Biblioteca de rede neural em Python, utilizada para a interface com implementações de rede neural (ex. TensorFlow).
Matplotlib	3.2.2	Biblioteca do Python que permite geração de histogramas e outros gráficos 2D a partir de um conjunto de dados
NLTK(*Natural Language Toolkit*)	3.6.2	Conjunto de bibliotecas e programas para processamento simbólico e estatístico da linguagem natural
NumPy	1.21.6	Biblioteca do Python que permite o fácil manuseio de vetores e matrizes
Pandas	1.2.5	Biblioteca de software criada para a linguagem Python com a finalidade de fazer manipulação e análise de dados
Python	3.9.5	Linguagem de programação
Re	3.9.5	Biblioteca Python nativa que permitiu a manipulação de texto por meio de expressões regulares
Scikit-learn	0.24	Biblioteca de aprendizado de máquina para a linguagem de programação Python
Visual Paradigm	16.3	Ferramenta para modelagem do sistema

Fonte: Autoria própria.

- NLTK (BIRD; LOPER, 2004): esta biblioteca é centrada na implementação de tarefas de PLN, fornecendo ferramentas para *tokenização*, realizada na etapa de pré-processamento textual deste trabalho, dentre outras.

- NumPy (HARRIS *et al.*, 2020): biblioteca do Python utilizada para manipulação de vetores, permitiu o fácil manuseio de vetores no desenvolvimento dos códigos deste trabalho.
- Pandas (MCKINNEY, 2011): esta biblioteca do Python possui métodos simples e eficazes de manipulação e visualização de dados que foram úteis desde a importação dos dados à realização das etapas de pré-processamento.
- Re: abreviação para *regular expressions*, ou expressões regulares em Português, esta biblioteca nativa do Python permitiu a manipulação de texto com expressões regulares.
- Scikit-learn (PEDREGOSA *et al.*, 2012): uma biblioteca que permite instanciar e utilizar múltiplos modelos de algoritmos de aprendizado de máquina e modelagem estatística, dentre ele algoritmos de classificação, regressão e *clustering*. Ela permitiu a instanciação e o uso de algoritmos de regressão, que foram utilizados nas etapas de treinamento do modelo e predição.

Quadro 3 – Especificações do computador utilizado para treinamento do modelo de inferência

Processador	CPU de 3,30GHz, 64 bits, 20 núcleos de processamento e tecnologia de virtualização
Memória RAM	31,0 GB
Sistema Operacional	Linux x86_64 20.04.1

Fonte: Autoria própria.

O Visual Paradigm é uma ferramenta que permite modelar diferentes tipos de diagramas populares de engenharia de software de forma rápida e eficiente, dentre eles os diagramas dinâmicos da UML (Linguagem de Modelagem Unificada, do Inglês *Unified Modeling Language*) (ex.atividade e sequência), os quais foram utilizados neste trabalho para propiciar um maior entendimento sobre as etapas da metodologia.

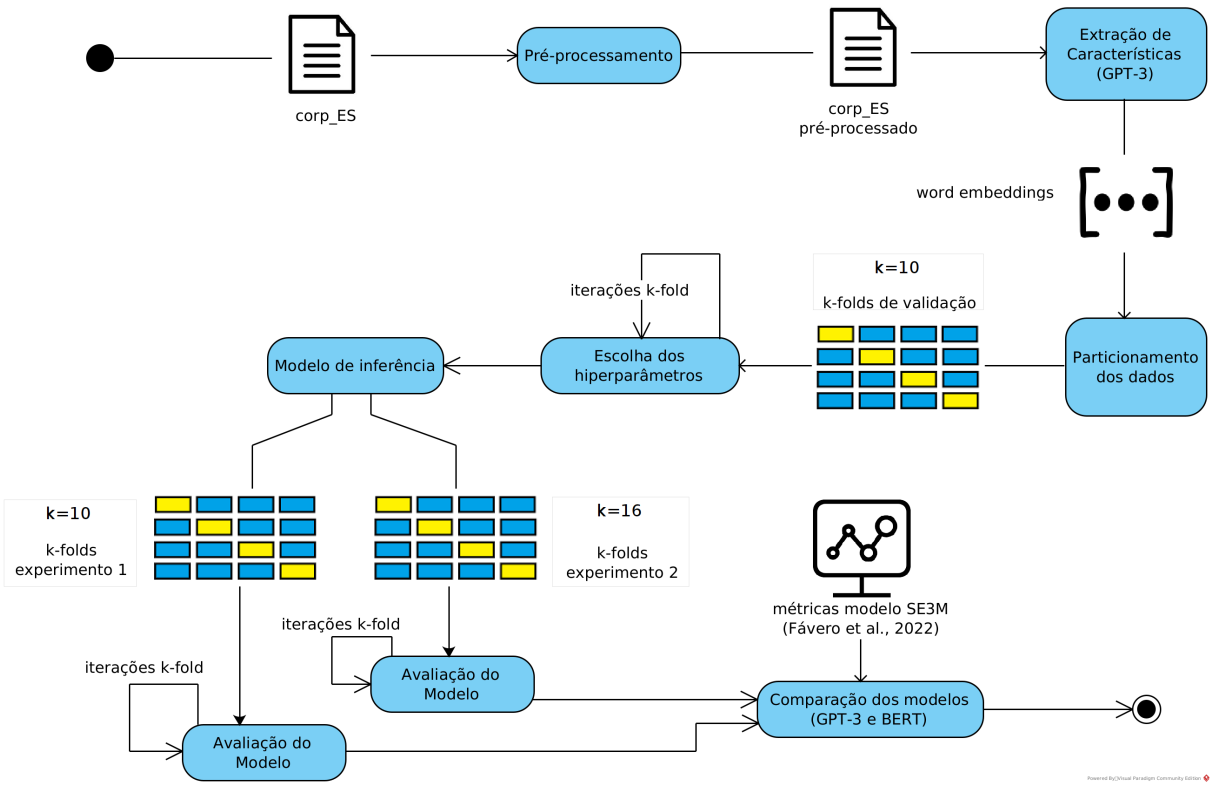
O modelo pré-treinado utilizado para etapa de representação textual foi o GPT-3 (BROWN *et al.*, 2020), um modelo de DL que utiliza o decodificador do mecanismo de autoatenção *Transformer* (VASWANI *et al.*, 2017). Conforme mencionado anteriormente, este mecanismo permite que o modelo se concentre seletivamente em segmentos de texto de entrada que ele prevê serem os mais relevantes para um dado contexto (JURAFSKY; MARTIN, 2019). Outra vantagem deste modelo, é o fato de ele ser pré-treinado em uma grande coleção de textos de diversas aplicações, de forma a oferecer um bom desempenho na geração de *word embeddings* mesmo sem a realização de um ajuste-fino (BROWN *et al.*, 2020). Uma explicação mais detalhada do GPT-3 e sua família pode ser encontrada na Subseção 2.4.4.2.

3.2 Método

Esse trabalho objetiva aplicar métodos de PLN e AM para gerar estimativas de esforço de software por regressão. Desta forma, pretende estimar o esforço necessário para a implementação de um requisito de software, com base em textos de requisitos (ex. histórias de usuário). Mais especificamente, esse trabalho pretende realizar uma comparação de desempenho entre dois modelos pré-treinados contextualizados: BERT e GPT-3, quando aplicados para essa finalidade. Sendo assim, foi tomado como base para a realização desse projeto, o estudo realizado por Fávero, Casanova e Pimentel (2022), em que foi proposto um modelo para EESA baseado em textos de histórias de usuário, fazendo uso do modelo BERT. Para realização da estimativa, o modelo utilizou como referência textos de histórias de usuário rotulados com a quantidade real de tempo necessária para a implementação de cada tarefa (dado em pontos por história). Com isso, o objetivo é encontrar itens textuais de requisitos similares e, basear-se em suas respectivas estimativas, para assim gerar uma estimativa para um novo requisito.

A seguir são apresentadas as etapas da metodologia adotada para o desenvolvimento deste trabalho, tais como: pré-processamento, preparação e particionamento dos dados, representação textual (extração de características), seleção do método de inferência, obtenção das estimativas e avaliação do modelo. A Figura 12 fornece uma visão geral destas etapas.

Figura 12 – Metodologia proposta



Fonte: Autoria própria.

3.2.1 Coleta de dados e Pré-processamento

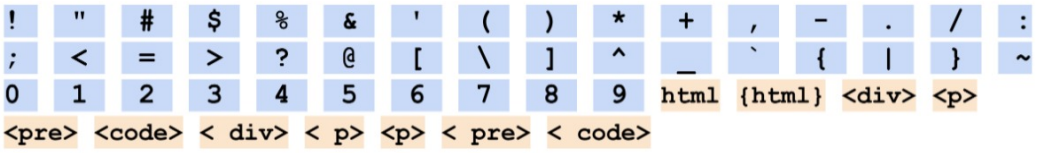
Para a formação do conjunto de dados de treinamento e testes, foi decidido pela utilização da mesma base de dados de textos de requisitos de software empregada pelo modelo SE^{3}M (FÁVERO; CASANOVA; PIMENTEL, 2022), a qual é a mesma aplicada por Choetkiertikul *et al.* (2019) em seu trabalho de pesquisa que objetiva obter estimativas de esforço de software a partir de textos de requisitos. Desta forma foi possível realizar uma comparação posterior das métricas de desempenho obtidas com o modelo proposto às obtidas pelo modelo SE^{3}M.

Sendo assim, para as etapas de treinamento e testes foi aplicado o corpus de texto aqui denominado *corp_ES* (CHOETKIERTIKUL *et al.*, 2019; FÁVERO; CASANOVA; PIMENTEL, 2022), elaborado especificamente para o contexto de engenharia de software. Esse corpus textual é composto de requisitos textuais na forma de histórias de usuário, sendo que o *corp_ES* possui os rótulos de esforço de desenvolvimento para cada história de usuário em *story points*[1]. É importante mencionar que os requisitos textuais desta base encontram-se em Inglês, o que limita o escopo da utilização do modelo de inferência proposto neste trabalho, para requisitos textuais nesta linguagem.

O *corp_ES* precisou passar por métodos de pré-processamento a fim de extrair alguns ruídos existentes no texto, sem comprometer o contexto das sentenças. Procurando manter uma comparação direta e mais justa possível com o modelo SE^{3}M (FÁVERO; CASANOVA; PIMENTEL, 2022), foi aplicado o mesmo método de pré-processamento. Desta forma, foram realizadas a remoção de caracteres especiais (ex. '{', '}', '#') e números, e também a conversão de todo o texto para o formato minúsculo. Em seguida, realizou-se a *tokenização* dos requisitos.

A Figura 13 lista a *stoplist* aplicada, ou seja, a lista dos caracteres e dos termos que foram removidos do texto. Nesta figura, os caracteres especiais e dígitos removidos estão destacados em azul claro, enquanto as *tags* HTML removidas estão destacadas com a cor laranja.

Figura 13 – Lista de caracteres e termos removidos do texto dos requisitos (*stoplist*). Em azul destacam-se os caracteres especiais e os dígitos, enquanto em laranja destacam-se as *tags html* removidos.



Fonte: Autoria própria.

[1] Os *story points* (ou pontos de história, em tradução literal do Inglês). Trata-se de uma unidade de medida para expressar uma estimativa do esforço geral necessário para implementar totalmente um item de um produto ou trabalho. Eles são estimados com base na complexidade do trabalho, quantidade de trabalho e risco ou incerteza (LEFFINGWELL, 2010)

3.2.2 Extração de características/representação textual

Para a extração de características dos textos de requisitos, transformando-os em vetores de *embeddings*, foi utilizado o modelo pré-treinado GPT-3 (BROWN *et al.*, 2020), por meio de sua API em Python, disponibilizada pela empresa OpenAI.

Sendo assim, com os textos de requisitos pré-processados, ainda foi necessário realizar a adequação dos textos para a entrada da API, bem como a configuração de alguns parâmetros, conforme especificado na documentação da OpenAI.

É importante mencionar que o uso da API do GPT-3 é parcialmente gratuito: ao menos no período do desenvolvimento deste trabalho, seu uso era cobrado por número de *tokens* fornecidos como entrada para o modelo e a OpenAI permite o uso gratuito de certa uma certa quantia de *tokens* por um determinado período. Levando isto em conta, foi utilizado o modelo *Babbage* do GPT-3, mais especificamente o modelo *text-similarity-babbage-001*, que é específico para o reconhecimento de semelhanças semânticas entre sentenças de texto. Esse modelo possui 2048 dimensões e suporta até 2048 *tokens* de entrada por requisição. Este modelo foi escolhido pela possibilidade de obter um bom desempenho na extração de características, ao mesmo tempo que por um custo relativamente baixo permite o envio de um volume significativo de *tokens*.

3.2.3 Métodos de Particionamento

O método utilizado para o particionamento do conjunto de dados (*corp_ES*) nas etapas de treinamento, validação e avaliação do modelo foi o *k-fold* aninhado (ver Subseção 2.5.4). Deste modo, na divisão dos subconjuntos de dados para o laço externo do método e, portanto, na escolha do valor do k utilizado neste laço, foram realizados os experimentos **E1** e **E2**, conforme segue:

- **E1** (multi repositórios): utilizando a divisão original do método *k-fold* aninhado, com um número de subconjuntos do laço externo após o particionamento k_1 igual a 10. Deste modo, este experimento é dito multi repositórios (do Inglês *multi-repository*), uma vez que os conjuntos de treino e teste possuem amostras de requisitos de mais de um repositório.

- **E2** (entre repositórios): utilizando particionamento por projetos, de modo que o número de subconjuntos do laço externo após o particionamento k_1 foi 16, uma vez o corpus de texto *corp_ES* é composto por 16 projetos. Já no caso deste experimento, diz-se que ele é entre repositórios (do Inglês *cross-repository*), uma vez que o conjunto de teste é composto por elementos de um único repositório.

Importante destacar que no experimento E1, cada subconjunto do laço externo possuía um número aproximadamente igual de elementos (requisitos textuais), elementos estes que foram embaralhados e distribuídos entre os subconjuntos, enquanto no experimento E2, o número de elementos em cada subconjunto do laço externo foi equivalente ao número de requisitos textuais de seu respectivo repositório, conforme ilustrado no Tabela 1.

Além disso, tanto em E1 quanto em E2, a escolha do número de subconjuntos do laço externo k_1 levou em conta a recomendação dos autores Breiman e Spector (1992) e Kohavi (2001) para um número de amostras N, maior que 160. E ainda, deve-se ressaltar que as divisões utilizadas em ambos os experimentos também foram testadas por Fávero, Casanova e Pimentel (2022) em dois de seus experimentos no desenvolvimento do SE^{3}M.

Já para o laço interno do *k-fold* aninhado, que realizou a escolha dos melhores hiperparâmetros, em ambos os experimentos (E1 e E2) foi utilizado um número de subconjuntos k_2 igual a 10, por conta do corpus textual *corp_ES* possuir relativamente poucos dados, necessitando assim de conjuntos maiores de treino para obter melhor desempenho.

Finalmente, em ambos os experimentos, após a divisão em subconjuntos e o treinamento com cada uma das diferentes combinações, foi calculada a média e seu desvio padrão entre as diferentes métricas obtidas.

Tabela 1 – Número de requisitos textuais (histórias de usuário) e descrição dos projetos utilizados nos experimentos

Número do projeto	Descrição	Requisitos/projeto
0	Mesos	1680
1	Usergrid	482
2	Appcelerator Studio	2919
3	Aptanastudio	829
4	Titanium	2251
5	DuraCloud	666
6	Bamboo	521
7	Clover	384
8	JIRA Software	352
9	Moodle	1166
10	Data Management	4667
11	Mule	889
12	Mule Studio	732
13	Spring XD	3526
14	Talend Data Quality	1381
15	Talend ESB	868
Total		**23.313**

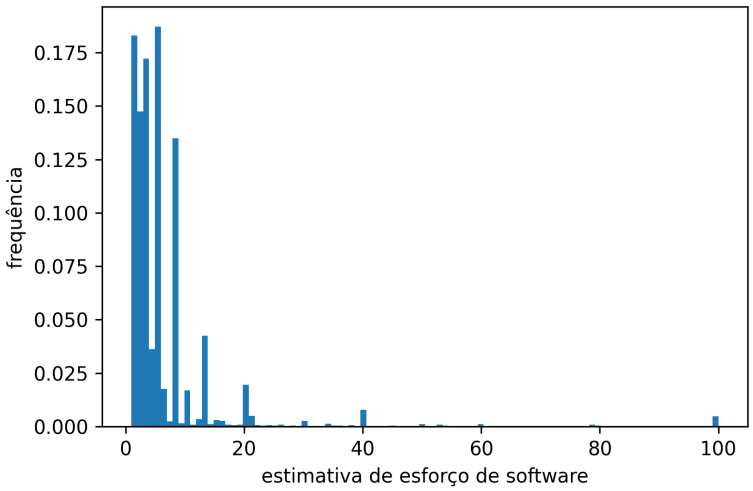
Fonte: Choetkiertikul *et al.* (2019)

3.2.4 Método de Inferência

Os rótulos correspondentes do esforço de software vinculados a cada história de usuário disponível no *dataset*, são representados de forma numérica e por valores contínuos que variam de 1 a 100, apesar de alguns valores estarem ausentes, como é possível observar no

histograma da Figura 14. Desta forma, a estimativa foi tratada como um problema de regressão, tal qual foi assumido no modelo SE^{3}M (FÁVERO; CASANOVA; PIMENTEL, 2022).

Figura 14 – Histograma representando a medida do esforço em relação à sua frequência no *corp_-ES*.



Fonte: Autoria Própria.

Desta forma, na saída do modelo de inferência, foi utilizado um modelo de regressão simples, por se tratar de uma única saída (a EESA). Para conseguir lidar com possíveis não linearidades no mapeamento dos *embeddings* para o valor da EESA, utilizou-se um modelo de Redes Neurais Artificiais (ANN), da mesma forma que foi utilizado no modelo de Choetkiertikul *et al.* (2019) e no SE^{3}M (FÁVERO; CASANOVA; PIMENTEL, 2022). Entretanto, diferente dos modelos de inferência empregados nos trabalhos citados, que utilizaram uma combinação de camadas do tipo densas com uma camada LSTM em suas redes neurais, este modelo de inferência utilizou apenas camadas densas, uma vez que com o modelo GPT-3, é gerado apenas um vetor de *embedding* para cada sentença (ou requisito) de entrada, e não um vetor para cada palavra como nos modelos de representação utilizados naqueles trabalhos, eliminando a dependência temporal entre as entradas da rede.

Assim, a rede neural utilizada foi do tipo *feedforward* (FNN), e utilizou cinco camadas, sendo uma camada de entrada, três camadas ocultas (densas), e uma camada de saída (densa). A camada de entrada recebeu como entrada os vetores de *embeddings* dos requisitos, com 2048 características diferentes. Em cada camada oculta foram utilizados 512 neurônios com funções de ativação ReLU, enquanto na camada de saída foi utilizado 1 neurônio com função de ativação linear (para realização da regressão). O número de camadas ocultas foi retirado do modelo SE^{3}M (FÁVERO; CASANOVA; PIMENTEL, 2022), enquanto o número de nós por camada escolhido de modo a balancear o tempo de execução e precisão do treinamento do modelo. A arquitetura da rede neural do modelo de inferência é ilustrada pelo Quadro 4.

Quadro 4 – Arquitetura da rede neural do modelo de inferência.

Camada (tipo)	Dimensão de saída	Núm. de Paramâmetros
entrada_1 (Entrada)	2048	0
densa_1 (Densa)	512	1049088
densa_2 (Densa)	512	262656
densa_3 (Densa)	512	262656
densa_4 (Densa)	1	513

Fonte: Autoria própria.

Para a escolha dos hiperparâmetros do modelo, foi realizada uma busca por *grid search*, que testou várias combinações possíveis dos mesmos exaustivamente. Deste modo, em cada iteração do laço externo do *k-fold*, o *grid search* selecionou o hiperparâmetro que apresentou o menor erro médio (durante suas iterações) no laço interno do *k-fold*. Ainda, o laço interno (tanto do *grid search* como do *k-fold*) utilizado pela busca de hiperparâmetros, utilizou o método de validação cruzada *k-fold* com um k igual a 10 (conforme mencionado na Subseção 3.2.3).

Para o treinamento do modelo de inferência, foram utilizadas 48 épocas e um mecanismo de *early stopping*, na qual, caso o valor do MAE de validação permanecesse estável por 10 épocas, o melhor resultado era restaurado. A variável MSE de validação foi escolhida como a função de perda. O tamanho do *batch_size* foi incluído na busca do *grid search*. Além disso, foi utilizado o otimizador Adam (KINGMA; BA, 2014), de modo que sua taxa de aprendizado e seus parâmetros β_1 e β_2 também foram experimentados pelo *grid search*. Os hiperparâmetros testados são listados pelo Quadro 5. Deste quadro, o hiperparâmetro "tamanho do *batch*" foi escolhido de modo a ser menor que o que foi utilizado no SE^{3}M, enquanto os outros hiperparâmetros foram escolhidos observando os valores recomendados para os mesmos no artigo do otimizador (KINGMA; BA, 2014).

Quadro 5 – Hiperparâmetros testados pelo *grid search*

hiperparâmetro	taxa de aprendizado (otimizador)	tamanho do *batch* (treinamento)	β_1 (otimizador)	β_2 (otimizador)
valores testados	$1e{-2}$	128	$0{,}99$	$0{,}999$
	$1e{-3}$	64	$0{,}95$	$0{,}99$
	$1e{-4}$		$0{,}90$	$0{,}9$
	$1e{-5}$			

Fonte: Autoria própria.

3.2.5 Avaliação

A partir do treinamento do modelo de inferência, foram calculadas algumas métricas para possibilitar uma avaliação da precisão das estimativas em cima dos conjuntos de treino.

Para cada particionamento do laço *k-fold* externo (em ambos os experimentos), as métricas utilizadas foram o MAE, o MdAE e o MSE, de modo que estas foram calculadas com as estimativas geradas e reais de cada requisito de teste. Ao final do laço *k-fold* externo, fo-

ram calculadas as médias e seu desvio padrão para cada métricas. Além disso, também foi determinado um intervalo de confiança de 95% para cada uma destas métricas.

Com as métricas de avaliação calculadas, foi realizada uma comparação das mesmas com as respectivas métricas obtidas no modelo SE^{3}M (FÁVERO; CASANOVA; PIMENTEL, 2022), a fim de comparar o desempenho dos modelos GPT-3 e BERT na tarefa de EESA baseado em textos. Além disso, compararam-se os resultados do experimento E1 também com o modelo Deep-SE (CHOETKIERTIKUL *et al.*, 2019), aproveitando que ele utilizou o mesmo corpus textual (Corp_ES).

O próximo capítulo apresenta os resultados do desenvolvimento da proposta aqui apresentada, bem como discussões pertinentes a eles.

4 RESULTADOS

A seguir são apresentados os resultados obtidos após a realização de cada um dos objetivos específicos propostos inicialmente. Ao final, é apresentada uma discussão dos resultados obtidos, objetivando responder à questão de pesquisa apresentada, verificando assim, a hipótese de pesquisa.

4.1 Preparação do Dataset

Primeiramente, foi realizado o pré-processamento sobre o corpus textual (*corp_ES*), o mesmo utilizado por Choetkiertikul *et al.* (2019) e Fávero, Casanova e Pimentel (2022). Para se ter conhecimento do antes e depois da realização do pré-processamento nos requisitos textuais, foram criados os Quadros 6 e 7 respectivamente, ambos contando com a mesma amostra de requisitos textuais, retirada desse corpus.

Quadro 6 – 5 amostras de requisitos da base *corp_ES* (ainda sem pré-processamento)

1	add ca against object literals in function invocations{html} <p>the idea here is that if our metadata captures a type as function arg, we should be able to create an instance of that type as an object literal as an arg to a function invocation. for example:< p> <pre> <code>ti.ui.createlabel({ <property-ca-here> });< code> < pre>< div>{html}
2	update branding for appcelerator plugin to appcelerator logo{html} <p>at least fix feature icons, associated natures. perhaps other screens as well.< p>< div>{html}
3	create new json schema for sdk team{html} <p>create json schema containing properties required for proper ca. hand schema to platform team to output new docs json files as part of new sdk releases. it may be that instead we can use the vsdoc format, in which case we will need to indicate to the sdk team additional items to be added to the docs.< p>< div>{html}
4	create project references property page{html} <p>create property page for project which allows manipulation of embedded references. user can add or remove references from the filesystem or url (if possible).< p> <p>there may well be an established metaphor or existing page in php, pdt, jsdt or elsewhere. investigate reusing this page is possible or appropriate.< p>< div>{html}
5	new desktop project wizard{html} <p>desktop (need to convert existing project creation code to java). some items here: <a href"https: github.com appcelerator titanium_developer blob master resources js project.js#l551» https: github.com appcelerator titanium_developer blob master reso...< a>. others here: <a href"https: github.com appcelerator titanium_developer blob master resources perspectives projects js projects.js» https: github.com appcelerator titanium_developer blob master reso...< a>.< p>< div>html

Fonte: Autoria própria.

Destaca-se ainda que o pré-processamento realizado removeu 2910864 caracteres de todo o corpus textual de requisitos, o que representa uma redução de cerca de 17,3% do número

Quadro 7 – 5 amostras de requisitos da base *corp_ES* após pré-processamento

1	add ca against object literals in function invocations the idea here is that if our metadata captures a type as function arg we should be able to create an instance of that type as an object literal as an arg to a function invocation for example tiuicreatelabel ltpropertycaheregt
2	update branding for appcelerator plugin to appcelerator logo at least fix feature icons associated natures perhaps other screens as well
3	create new json schema for sdk team create json schema containing properties required for proper ca hand schema to platform team to output new docs json files as part of new sdk releases it may be that instead we can use the vsdoc format in which case we will need to indicate to the sdk team additional items to be added to the docs
4	create project references property page create property page for project which allows manipulation of embedded references user can add or remove references from the filesystem or url if possible there may well be an established metaphor or existing page in php pdt jsdt or elsewhere investigate reusing this page is possible or appropriate
5	new desktop project wizard desktop need to convert existing project creation code to java some items here a href https github com appcelerator titaniumdeveloper blob master resources js projectjsl https github com appcelerator titaniumdeveloper blob master reso a others here a href https github com appcelerator titaniumdeveloper blob master resources perspectives projects js projectsjs https github com appcelerator titaniumdeveloper blob master reso a

Fonte: Autoria própria.

de caracteres em relação ao corpus sem pré-processamento. Essa porcentagem significa que, a cada 6 caracteres do corpus original, 1.04 foram removidos nesta etapa, aproximadamente.

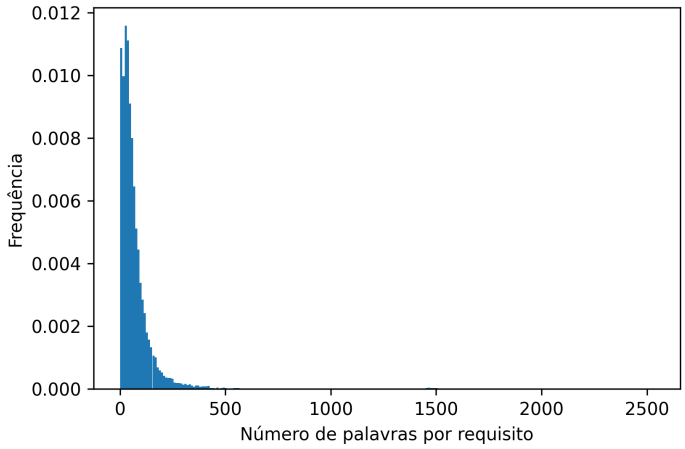
4.2 Representação Textual via GPT-3

Segundo a documentação da API do GPT-3, é necessário realizar os seguintes passos antes de utilizá-la (OPENAI, 2022?):

1. *tokenização* das palavras;
2. remoção de quebras de linha (caracteres "\n") no texto de entrada;
3. escolha de um dos diferentes modelos disponibilizados pela API (*Davinci*, *Curie*, *Babbage* ou *Ada*).
4. truncamento ou remoção de textos de entrada com mais de 2048 *tokens*.

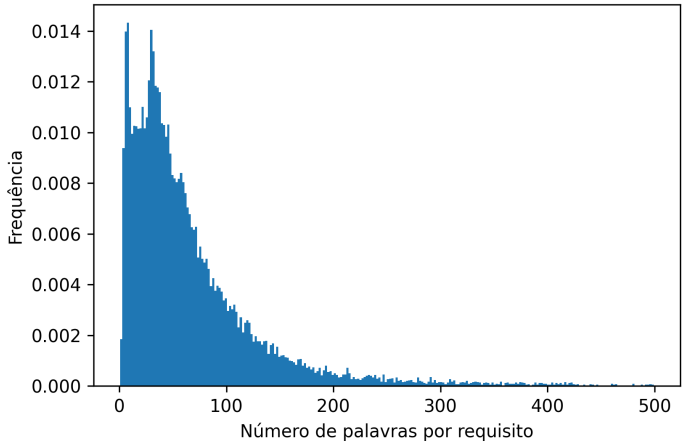
Para garantir que os requisitos não ultrapassassem 2048 *tokens*, foram truncados em 2000 *tokens* os requisitos que apresentaram mais que esse número, conforme a recomendação da documentação da API do GPT-3 (OPENAI, 2022?). Para se ter uma ideia da quantidade de requisitos que foram truncados com essa regra, o histograma da Figura 15 exibe uma distribuição do número de *tokens* por requisito. Para melhorar a visualização de forma gráfica,

Figura 15 – Histograma representando o número de palavras em cada sentença no *corp_ES*.



Fonte: Autoria Própria.

Figura 16 – Histograma representando o número de palavras em cada sentença no *corp_ES*, limitado a 500 palavras.



Fonte: Autoria Própria.

limitou-se o número de palavras exibidas no histograma da Figura 16 para quinhentas (500). Desta forma, percebe-se que a maior parte dos textos de requisitos (23125), cerca de 99.2% deles, apresentam até 500 *tokens*.

Seguidas as recomendações da API, os requisitos pré-processados foram enviados para a API do GPT-3, e obtiveram-se os vetores de *embeddings* para cada um deles. Assim, o Quadro 8 exibe os *word embeddings* para os respectivos requisitos do Quadro 7.

Ainda no Quadro 8, observa-se, que os vetores de *embeddings* obtidos com o modelo *Babbage* do GPT-3, possuem 2048 dimensões. Comparado com os modelos pré-treinados do BERT (DEVLIN *et al.*, 2018) (ex. modelo *BERT_base* utilizado por Fávero, Casanova e Pimentel (2022) e modelo *BERT_large*), cada um com 768 dimensões e 1024 dimensões de saída, respectivamente, o *Babbage* apresenta no mínimo o dobro de dimensões de saída. Isso pode

Quadro 8 – Representação por *word embeddings* obtidos para requisitos do *corp_ES*. Cada vetor exibido corresponde ao *word embedding* obtido para um requisito textual, e contém 2048 características (valores em decimal).

	0	1	2	...	2045	2046	2047
embedding requisito 1 →	0.0026	0.0216	-0.0109	...	-0.0105	0.0055	-0.0083
	0	1	2	...	2045	2046	2047
embedding requisito 2 →	-0.0046	0.0128	-0.0244	...	0.0309	-0.0303	-0.0002
	0	1	2	...	2045	2046	2047
embedding requisito 3 →	-0.0072	0.0176	0.0087	...	0.0142	-0.0296	-0.0195
	0	1	2	...	2045	2046	2047
embedding requisito 4 →	0.0041	0.0228	-0.0062	...	0.0215	-0.0027	-0.0021
	0	1	2	...	2045	2046	2047
embedding requisito 5 →	-0.0296	0.0246	-0.0070	...	-0.0210	-0.0146	-0.0270

Fonte: Autoria própria.

justificar-se porque o volume de dados usados para o pré-treino do GPT-3 foi muito maior e, consequentemente, o seu nível de aprendizado também. Em outras palavras, incorporar palavras em um espaço de alta dimensão requer um maior volume de dados para reforçar a densidade e o significado da representação, o que foi realizado com o GPT-3.

4.3 Métricas obtidas

Após a obtenção dos *embeddings* para todos os requisitos do corpus, foram realizados os experimentos E1 e E2 (ver Subseção 3.2.3), na qual diferentes combinações de hiperparâmetros foram testadas pelo método *k-fold* aninhado, e diferentes modelos de inferência foram treinados. Ao final dos dois experimentos, obteve-se a Tabela 2, que apresenta uma estimativa sem viés para as métricas MAE, MdAE e MSE, com intervalo de confiança de 95%, extraídas na avaliação dos modelos de inferência com os melhores hiperparâmetros de cada subconjunto do laço externo. Com esta tabela, é possível observar que o experimento E1, o qual realizou um particionamento homogêneo e randomizado dos subconjuntos do laço externo, obteve um desempenho bem adequado entre as diferentes etapas de avaliação. Após realizar a média e o desvio padrão das métricas obtidas, foi possível observar que, apesar de E1 ser multi-repositório, esse apresentou intervalos de confiança (e desvios padrão) bem mais restritos em comparação ao experimento E2. Isso indica que as estimativas obtidas no E1 são mais estáveis e a suas medidas de desempenho mais confiáveis do que as do E2.

Além disso, na Tabela 2, também observa-se que no experimento E2 as médias dos erros MAE, MSE e MdAE sobre suas estimativas também foram maiores do que os respectivos erros do experimento E1. Também, a média do desvio padrão obtido foi maior, de modo

Tabela 2 – Avaliação dos resultados obtidos paras os experimentos E1 e E2. Para ambas as métricas avaliadas, quanto menor for o valor, melhor o resultado.

Experimento	MAE	MSE	MdAE
E1	3,67 $\pm$ 0,12	64,38 $\pm$ 6,45	1,88
E2	3.80 $\pm$ 1.20	68.87 $\pm$ 65.93	2.26

Fonte: Autoria própria.

que é possível afirmar que o experimento E1 obteve um desempenho geral melhor do que o experimento E2.

A questão de pesquisa para esse trabalho objetivou saber se o modelo pré-treinado GPT-3 é tão eficiente quanto o modelo BERT para EESA a partir de requisitos textuais. Sendo assim, ao comparar os valores do MAE em cada subconjunto do experimento E2 com os valores obtidos em um experimento análogo realizado no modelo SE^{3}M (FÁVERO; CASANOVA; PIMENTEL, 2022), obteve-se a Tabela 3, e ao comparar os valores de MAE, MSE e MdAE do experimento E1 com os obtidos pelos modelos SE^{3}M e Deep-SE (CHOETKIERTIKUL *et al.*, 2019), obteve-se os resultados da Tabela 4. Na última tabela, entretanto, os valores de MSE e MdAE do modelo Deep-SE foram omitidos por conta de o mesmo não ter disponibilizado estas métricas em seu trabalho.

Tabela 3 – Valores do MAE obtidos em cada subconjunto no experimento E2 e do experimento análogo realizado no SE^{3}M, com dados dos respectivos projetos utilizados para avaliação do modelo. Também são apresentados o identificador do projeto (ID), o número de requisitos por projeto (Requisitos/projeto), a média (M_E) e o desvio padrão (DP) do esforço por requisito em cada projeto. Quanto menor o valor do MAE, melhor o resultado. Os menores valores de MAE de cada projeto foram destacados em negrito.

ID	Descrição	Requisitos/projeto	M_E	DP	MAE (GPT-3)	MAE (SE^{3}M)
AS	Appcelerator Studio	2919	5,63	3,32	**2,26**	2,50
AP	Aptanastudio	829	8,01	5,95	**4,05**	4,18
BB	Bamboo	521	2,41	2,14	**2,31**	2,76
CV	Clover	384	4,60	6,54	**3,54**	3,87
DM	Data Management	4667	9,56	16,6	**7,75**	7,78
DC	DuraCloud	666	2,12	2,03	**3,53**	3,79
JI	JIRA Software	352	4,43	3,51	4,75	**3,13**
ME	Mesos	1680	3,08	2,42	**1,54**	3,39
MD	Moodle	1166	15,54	21,63	**11,54**	11,99
MU	Mule	889	5,08	3,49	**2,65**	3,51
MS	Mule Studio	732	6,39	5,38	**3,42**	3,51
XD	Spring XD	3526	3,69	3,22	**2,27**	3,16
TD	Talend Data Quality	1381	5,92	5,19	**3,54**	4,04
TE	Talend ESB	868	2,16	1,49	**2,67**	3,42
TI	Titanium	2251	6,31	5,09	**3,24**	3,49
UG	Usergrid	482	2,85	1,40	**1,76**	3,24

Fonte: Adaptado de Fávero, Casanova e Pimentel (2022).

Na Tabela 3, nota-se que o experimento E2 obteve valores de MAE menores em 15 dos 16 subconjuntos de teste. Já na Tabela 4, é possível notar que o experimento E1, neste traba-

Tabela 4 – Erro Absoluto Médio (MAE), Erro Quadrático Médio (MSE) e Mediana dos Erros Absolutos (MdAE) obtidos ao aplicar o modelo GPT-3 (Z-SE^2), comparados aos modelos SE^{3}M (BERT) e Deep-SE (Word2Vec). Para todas as métricas, quanto menor o valor melhor o resultados.

Método	Z-SE^2 (multi repositórios)	SE^{3}M (multi repositórios)	Deep-SE (entre repositórios)
MAE	3,67 $\pm$ 0,12	4,25 $\pm$ 0,17	3,82 $\pm$ 1,56
MSE	64,38 $\pm$ 6,45	86,15 $\pm$ 1,66	-
MdAE	1,88	2,3	-

Fonte: Adaptado de Choetkiertikul *et al.* (2019), Fávero, Casanova e Pimentel (2022).

lho nomeado Z-SE^2 (Estimador de Esforço de Software Sem Ajuste-fino, do Inglês *Zero-shot Software Effort Estimator*), além de obter um MAE menor, também obteve um desvio padrão menor em relação ao SE^{3}M e ao Deep-SE. Ainda, em relação ao modelo SE^{3}M, tanto o valor do MSE como do MdAE do Z-SE^2 foram menores. Deste modo, é possível afirmar que o seu desempenho foi no mínimo similar ao dos demais modelos.

Um exemplo de aplicação prática de MAE na EESA indica que, se uma história de usuário que apresenta um esforço real de 7 pontos por história, o valor do esforço estimado pelo modelo Z-SE^2 poderia estar entre 3,33 e 10,67 pontos por história. Já se for feita a mesma comparação utilizando o modelo SE^{3}M, que aplicou o modelo BERT, a mesma estimativa poderia ficar entre 2,75 e 11,25 pontos por história.

Ademais, deve-se ressaltar que o modelo SE^{3}M utilizou o modelo BERT (contextualizado), e o modelo Deep-SE proposto por Choetkiertikul *et al.* (2019) aplicou a representação via Word2Vec (sem-contexto). Porém ambos, se utilizaram de uma camada LSTM na arquitetura do modelo de inferência. Ainda, todos os três modelos comparados usaram o mesmo corpus textual (Corp_ES).

4.4 Discussão dos resultados

No inicio do trabalho, foi levantada a seguinte questão de pesquisa: O modelo pré-treinado GPT-3 é tão eficiente quanto o modelo BERT para a EESA a partir de requisitos textuais?

Ao buscar responder à questão de pesquisa para esse trabalho, obteve-se claramente um resultado similar ao aplicar o modelo GPT-3 para a representação textual, antes de usá-la como entrada no modelo de aprendizado proposto. Com isso, pode-se dizer que a hipótese definida inicialmente foi acertada, confirmando também as premissas do GPT-3 de obter ótimos resultados em aplicações de PLN sem necessidade de ajuste-fino. Destaca-se que o fato de obter resultados similares aos obtidos pelos modelos comparados mesmo sem a realização do ajuste-fino do modelo de linguagem configura uma vantagem do modelo proposto neste trabalho.

Quanto ao menor erro médio do experimento E1 em relação ao experimento E2, o fato de E1 utilizar o particionamento *multi repositórios* para os dados de treino e teste, pode ter permitido ao modelo uma maior capacidade de generalização, uma vez que possibilitou ao modelo observar uma maior diversidade de entradas do que no caso do particionamento *entre repositórios*, comprovando o fato de que o modelo GPT-3 consegue generalizar, conseguindo representar os textos adequadamente, mesmo sem ajuste aos dados do domínio.

Com relação à confiabilidade dos resultados, cabe observar que o experimento E2 obteve um desvio padrão muito maior que o experimento E1, ou seja, o particionamento *entre repositórios* obteve alguns erros menores do que a média, e outros resultados ruins, com erro bem maior que a média. Como já comentado na Seção 4.3, isto torna o resultado do experimento E2 pouco confiável, uma vez que há grande variação de seu desempenho.

Já quanto ao desempenho no mínimo similar do experimento E1 em relação aos modelos comparados pela Tabela 4, os fatores que podem ter influenciado neste resultado são: um maior número de nós na rede neural do modelo de inferência, a otimização de hiperparâmetros (*k-fold* aninhado + *grid search*), e o uso do modelo de representação textual GPT-3.

Com relação ao primeiro fator que pode ter influenciado na similaridade do desempenho do modelo Z-SE^2 em relação aos demais, este trata-se do maior número de nós por camada (ou seja, maior profundidade da rede neural) do modelo proposto neste trabalho em relação às redes neurais utilizadas no SE^{3}M (FÁVERO; CASANOVA; PIMENTEL, 2022) e no Deep-SE (CHOETKIERTIKUL *et al.*, 2019). É possível que isto tenha influenciado, uma vez que quanto maior o corpus de texto, maior a influência da profundidade da rede no desempenho (MHASKAR; LIAO; POGGIO, 2017).

Outro fator que pode ter influenciado positivamente o resultado é o uso do *k-fold* aninhado em conjunto com o *grid search*. Com a combinação destes métodos, uma vez que a combinação de hiperparâmetros com o menor erro é escolhida para cada subconjunto da etapa de avaliação do modelo, pode-se obter um erro médio menor e uma estimativa mais precisa. No treinamento do modelo SE^{3}M não foi realizada a otimização de hiperparâmetros, e no modelo Deep-SE ela foi realizada, porém sem o uso do método de validação cruzada *k-fold*.

Finalmente, outro fator possivelmente responsável pelo desempenho similar do modelo usado neste trabalho, em relação aos modelos comparados, é uso do modelo de representação textual GPT-3 na extração dos *word embeddings* dos requisitos textuais. O modelo *Babbage* do GPT-3 possui 1,3 bilhões de parâmetros, cerca de 12 vezes o número de parâmetros que possui o BERT_base (110 milhões) utilizado no SE^{3}M. Além disso, em relação ao tamanho dos vetores de *embeddings*, o GPT-3 *Babbage* possui 2048 dimensões, contra 768 e 100 dos *embeddings* utilizados no SE^{3}M e no Deep-SE, respectivamente. Tanto o maior número de parâmetros do modelo de representação, quanto o tamanho da representação dos *embeddings* podem influenciar diretamente a qualidade da representação textual dos requisitos, o que pode também ter contribuído para o resultado similar (BROWN *et al.*, 2020; CHOETKIERTIKUL *et al.*, 2019; DEVLIN *et al.*, 2018; FÁVERO; CASANOVA; PIMENTEL, 2022).

Além disso, vale destacar que o desempenho nos experimentos E1 e E2 foram obtidos sem realização de ajuste-fino do modelo GPT-3 (conforme mencionado no início desta Seção), o que reforça a premissa de que ele oferece um ótimo desempenho mesmo sem o uso de qualquer ajuste-fino, assim como observado pelos seus autores (BROWN *et al.*, 2020). Diferente do modelo implementado neste trabalho, os modelos SE^{3}M e Deep-SE realizaram o ajuste-fino do seu modelo de representação textual, ou seja, os modelos foram pré-treinados novamente em um conjunto extenso de textos de requisitos de software, a fim de que o modelo gerasse aprendizado dentro de um contexto específico, neste caso da Engenharia de Software. Deste modo, a possibilidade de operar sem ajuste fino é mais uma vantagem no uso do modelo GPT-3 para a tarefa de EESA (CHOETKIERTIKUL *et al.*, 2019; FÁVERO; CASANOVA; PIMENTEL, 2022).

4.4.1 Possíveis Limitações e Melhorias

Alguns fatores podem ter limitado o desempenho do modelo proposto e, consequentemente, a obtenção de resultados similares deste trabalho em relação aos modelos comparados. O primeiro, e um dos mais importantes, se refere ao pré-processamento dos requisitos textuais que, apesar de remover caracteres indesejados, números e algumas *tags* HTML, não conseguiu remover por completo trechos de código e endereços da *web*, que compõem uma grande parcela das palavras presentes nos requisitos. Isso significa que, ao *tokenizar* os textos, endereços da *web*, por exemplo, tem seus pontos eliminados, restando palavras soltas, e que se tornam sem contexto.

O pré-processamento de textos com estas características encontradas em requisitos textuais, trata-se de uma tarefa complexa, e que requer um estudo mais aprofundado, aplicando expressões regulares, mas uma vez que não há um padrão entre onde começam e terminam os trechos de código, por exemplo, a limpeza do texto com esta técnica fica comprometida. Neste ponto, uma possível melhoria envolveria utilizar métodos inteligentes para remover estes endereços, trechos de código e outros elementos estranhos do texto, possivelmente aplicando representação por *embeddings* combinadas com métodos de Aprendizado de Máquina, por exemplo.

Ainda que o desempenho obtido pelos experimentos E1 e E2 ter se equiparado com o desempenho de outros trabalhos que utilizaram diferentes modelos de representação textual (ver Seção 4.3), na parte da representação textual, uma possível melhoria de desempenho poderia ser obtida com a realização do ajuste-fino do modelo GPT-3, conforme recomendação dos seus autores (BROWN *et al.*, 2020).

Entretanto, deve-se destacar que a realização do ajuste-fino nos modelos do GPT-3 possuem a limitação do custo elevado, que é cobrado por *tokens* e épocas utilizados do processo de ajuste-fino. Se o preço para utilizar o GPT-3 simplesmente para a geração de *embeddings* pode se tornar elevado para muitas aplicações, o preço para realizar o ajuste-fino do modelo e o

utilizar em seguida pode se tornar múltiplas vezes maior. Além disso, a realização do ajuste-fino para o caso específico da geração de *word embeddings* não é descrita na documentação do GPT-3, de modo que talvez não seja possível a realização do mesmo para a tarefa de geração de *embedding* neste modelo (OPENAI, 2022?).

Além disso, utilizar um modelo mais robusto do GPT-3, como por exemplo o *Curie* ou o *Davinci* ao invés do *Babbage*, também poderia melhorar o resultado, uma vez que estes modelos possuem um número maior de parâmetros (6,7 bilhões e 175 bilhões, respectivamente) e um número maior de dimensões de saída (4096 e 12288, respectivamente).

Quanto ao modelo de inferência, poucos hiperparâmetros testados e o próprio uso do *grid search* como método de otimização foram limitações para o desempenho do modelo deste trabalho. Uma vez que não se tinha uma ideia inicial de quais hiperparâmetros poderiam gerar melhores resultados, e também poucos hiperparâmetros foram testados pelo *grid search*, é pouco provável que os melhores hiperparâmetros locais, encontrados pelo método, estejam próximos dos melhores hiperparâmetros globais. Além disso, apesar de ser simples e de fácil utilização, o *grid search* não é realizado por padrão de forma distribuída, de modo que sua execução se torna demorada, mesmo testando poucos hiperparâmetros. Vale ressaltar também, que foram testados um pequeno número de hiperparâmetros de modo a limitar o custo computacional do *grid search*, que cresce exponencialmente à medida que novas variações de hiperparâmetros são adicionados.

Deste modo, o uso de um método de otimização de hiperparâmetros que conseguisse, de forma eficiente, testar várias combinações arbitrárias, como por exemplo métodos de otimização baseados em *early-stopping* (ver Subseção 2.5.5), poderia auxiliar na obtenção de melhores resultados com o modelo de inferência.

Ainda, no modelo de inferência, o uso de uma arquitetura de rede neural com um número maior de nós por camada (e assim, maior profundidade), outro número de camadas, e/ou outra combinação de tipos de camadas, também poderiam tornar o resultado ainda melhor. Mas, neste caso, a comparação com o modelo SE^3M tornaria-se menos justa, uma vez que a arquitetura da rede neural utilizada por ele seria ainda mais discrepante em relação a arquitetura da rede neural utilizada trabalho, havendo a necessidade de atualizá-lo com a nova arquitetura.

Por fim, outro fator limitante para o desempenho do modelo de inferência está no número de requisitos (23313) e no número de projetos (16) utilizados. Neste ponto, para aumentar a capacidade de generalização do modelo, seria necessário treinar a rede neural em uma maior variedade de projetos e um maior número de requisitos textuais, de modo que ela tornasse-se capaz de reconhecer características em um número maior de contextos.

Ressalta-se que a não utilização de testes estatísticos mais robustos como teste de hipóteses ou teste T limitou a possibilidade da comparação deste trabalho de verificar a superioridade dos resultados obtidos em relação aos resultados dos modelos comparados, uma vez que seria necessária a replicação dos resultados obtidos nestes outros modelos para assegurar-se do melhor resultado do modelo deste trabalho em relação aos outros comparados.

5 CONCLUSÃO

Este trabalho apresenta, como objetivo principal comparar o desempenho da aplicação dos modelos pré-treinados contextualizados GPT-3 e BERT – esse último empregado no SE^{3}M (FÁVERO; CASANOVA; PIMENTEL, 2022) – na representação de características textuais para a inferência de EESA.

Sendo assim, um dos objetivos foi a preparação da base de dados para a geração da representação textual por meio de *embeddings* com o GPT-3. Nesta etapa, a utilização de expressões regulares para a realização do pré-processamento dos requisitos foi uma limitação para este trabalho, uma vez que, apesar de conseguir remover certos caracteres e palavras indesejados, elas não foram suficientes para a remoção de trechos de código e nomes de domínios de sites nos requisitos, de modo que estes configuraram um ruído para a extração de características textuais.

Na etapa da extração de características textuais, uma limitação para o trabalho está no valor cobrado pela utilização do modelo pré-treinado GPT-3, uma vez que este é contabilizado pelo número de *tokens* fornecidos como entrada para o modelo e, em especial para a geração de *embeddings*. O valor é relativamente elevado, principalmente para aplicações que precisam gerar representações textuais para vários *tokens*, como no caso deste trabalho. É fato que o GPT-3 é conhecido por funcionar muito bem mesmo sem a realização do ajuste-fino, porém vale ressaltar que a realização do ajuste-fino do mesmo também é cobrado, representando mais uma limitação para aplicações com muitos *tokens* e orçamento pequeno. Apesar disso, o fato deste modelo não necessitar ajuste-fino pode ser considerado uma vantagem, pois permite a utilização direta do modelo.

Após a etapa de representação textual, foram aplicados métodos de AM para inferência de EESA e realizou-se um comparativo das métricas obtidas com as métricas obtidas pelo modelo BERT utilizado no SE^{3}M. Nesta etapa, a utilização da biblioteca Keras (CHOLLET *et al.*, 2015) para o treinamento do modelo de inferência configurou uma vantagem por ser simples e de fácil uso. Entretanto, por padrão esta biblioteca não realiza o treinamento de modelos de forma distribuída, tornando esta etapa lenta.

Conforme comentado anteriormente, a realização da otimização de hiperparâmetros com o método *grid search*, apesar de ser simples e de fácil utilização, também limitou este trabalho, de modo que poucos hiperparâmetros puderam ser testados. Além disso, sua execução não é realizada de forma distribuída.

Finalmente, apesar das limitações apresentadas, foi possível realizar a comparação do desempenho do modelo GPT-3 na inferência de EESA com o desempenho do modelo BERT utilizado no modelo SE^{3}M. Por meio desta comparação, comprovou-se a similaridade do primeiro modelo em relação ao segundo, validando a hipótese levantada no início do trabalho para a questão de pesquisa, destacando-se o diferencial da não realização do ajuste-fino.

Algumas hipóteses de possíveis fatores que podem ter influenciado nos resultados obtidos neste trabalho foram apresentados, apesar de não ser possível diferenciar, com certeza, quais destes realmente tiveram um impacto positivo e quais tiveram pouco ou nenhum impacto, o que configura a necessidade de realização de outros experimentos futuros.

Deste modo, listam-se algumas sugestões para trabalhos futuros envolvendo o modelo Z-SE^2:

- Para conseguir avaliar de forma mais confiável se o desempenho do modelo GPT-3 foi realmente superior do que o modelo BERT, quando aplicados à tarefa de EESA, sugere-se a utilização de testes estatísticos mais robustos como o teste de hipóteses ou o teste T, por exemplo.

- Para o pré-processamento, sugere-se a utilização de alguma técnica inteligente, fazendo uso de IA para detectar e remover endereços da web e trechos de código de forma autônoma e acertiva.

- Como alternativas gratuitas de código aberto para o modelo GPT-3, sugere-se o uso de modelos recentes como o OPT (do Inglês *Open Pre-trained Transformer*) (ZHANG *et al.*, 2022) ou o GPT-NeoX-20B (BLACK *et al.*, 2022), por exemplo. De preferência, recomenda-se o uso de um modelo de *word embeddings* contextualizado com um núMeFro próximo ou superior de parâmetros em relação ao GPT-3 *Babbage*, de modo a possibilitar a obtenção de melhores resultados.

- Como alternativa ao método *grid search* para a otimização de hiperparâmetros, recomenda-se o uso de métodos baseados em *early-stopping*, de modo a parar rapidamente o treinamento de hiperparâmetros ruins e assim conseguir testar uma maior variedade de hiperparâmetros.

- Quanto à biblioteca para o treinamento do modelo de inferência, recomenda-se utilizar o Keras e adaptar o código de modo a paralelizar sua execução, ou utilizar alguma outra biblioteca que realize a paralelização do treinamento de modelos de redes neurais, de modo a diminuir o tempo necessário para o treinamento do modelo.

REFERÊNCIAS

ABDUKALYKOV, R. **A New Methodology for Quantifying the Impact of Non-Functional Requirements on Software Effort Estimation**. 2011. Tese (Doutorado) — Concordia University, 2011.

ABRAHAMSSON, P. *et al.* Predicting development effort from user stories. *In*: **Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement**. IEEE Computer Society, 2011. p. 400–403. ISBN 9780769546049. Disponível em: https://doi.org/10.1109/ESEM.2011.58.

AHAD, M. A. R.; Das Antar, A.; AHMED, M. **IoT Sensor-Based Activity Recognition: Human Activity Recognition**. Springer International Publishing, 2020. (Intelligent Systems Reference Library). ISBN 9783030513795. Disponível em: https://books.google.com.br/books?id=tl70DwAAQBAJ.

AKBIK, A.; BERGMANN, T.; VOLLGRAF, R. Pooled Contextualized Embeddings for Named Entity Recognition. *In*: **Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)**. Minneapolis, Minnesota: Association for Computational Linguistics, 2019. p. 724–728. Disponível em: https://aclanthology.org/N19-1078.

ALAMMAR, J. Blog, **The Illustrated Transformer**. 2018. Disponível em: https://jalammar.github.io/illustrated-transformer/.

ALAMMAR, J. **The Illustrated GPT-2 (Visualizing Transformer Language Models)**. 2019. Disponível em: https://jalammar.github.io/illustrated-gpt2/.

AYYILDIZ, T. E.; KOÇYIGIT, A. A case study on the utilization of problem and solution domain measures for software size estimation. *In*: **2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)**. Limassol, Cyprus: IEEE, 2016. p. 108–111. Disponível em: http://dx.doi.org/10.1109/SEAA.2016.13.

BAHDANAU, D.; CHO, K. H.; BENGIO, Y. Neural machine translation by jointly learning to align and translate. **3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings**, International Conference on Learning Representations, ICLR, 9 2014. Disponível em: https://arxiv.org/abs/1409.0473v7.

BARUA, A. *et al.* Analysis of Contextual and Non-contextual Word Embedding Models for Hindi NER with Web Application for Data Collection. *In*: GARG, D. *et al.* (Ed.). **Advanced Computing**. Singapore: Springer Singapore, 2021. p. 183–202. ISBN 978-981-16-0401-0.

BENGIO, Y.; COURVILLE, A.; VINCENT, P. Representation learning: A review and new perspectives. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, IEEE, v. 35, n. 8, p. 1798–1828, 2013. Disponível em: https://doi.org/10.1109/TPAMI.2013.50.

BENGIO, Y.; SIMARD, P.; FRASCONI, P. Learning long-term dependencies with gradient descent is difficult. **IEEE transactions on neural networks**, IEEE Trans Neural Netw, v. 5, p. 157–166, 1994. ISSN 1045-9227. Disponível em: https://pubmed.ncbi.nlm.nih.gov/18267787/.

BENKE, K.; BENKE, G. Artificial Intelligence and Big Data in Public Health. **International Journal of Environmental Research and Public Health**, v. 15, n. 12, 2018. ISSN 1660-4601. Disponível em: https://www.mdpi.com/1660-4601/15/12/2796.

BERGSTRA, J.; BENGIO, Y. Random search for hyper-parameter optimization. **Journal of Machine Learning Research**, JMLR.org, v. 13, p. 281–305, 2 2012. ISSN 1532-4435. Disponível em: https://dl.acm.org/doi/10.5555/2188385.2188395.

BIRCANOĞLU, C.; ARICA, N. A comparison of activation functions in artificial neural networks. *In*: **2018 26th Signal Processing and Communications Applications Conference (SIU)**. Izmir, Turkey: IEEE, 2018. p. 1–4. Disponível em: https://doi.org/10.1109/SIU.2018.8404724.

BIRD, S.; LOPER, E. NLTK: The Natural Language Toolkit. *In*: **Proceedings of the ACL Interactive Poster and Demonstration Sessions**. Barcelona, Spain: Association for Computational Linguistics, 2004. p. 214–217. Disponível em: https://aclanthology.org/P04-3031.

BLACK, S. *et al.* Gpt-neox-20b: An open-source autoregressive language model. 4 2022. Disponível em: http://arxiv.org/abs/2204.06745.

BOEHM, B.; ABTS, C.; CHULANI, S. Software development cost estimation approaches—a survey. **Annals of software engineering**, Springer, v. 10, n. 1-4, p. 177–205, 2000. Disponível em: https://doi.org/10.1023/A:1018991717352.

BREIMAN, L.; SPECTOR, P. Submodel Selection and Evaluation in Regression. The X-Random Case. **International Statistical Review / Revue Internationale de Statistique**, International Statistical Institute (ISI), v. 60, n. 3, p. 291–319, 1992. ISSN 03067734, 17515823. Disponível em: http://www.jstor.org/stable/1403680.

BROWN, T. B. *et al.* Language models are few-shot learners. *In*: **Proceedings of the 34th International Conference on Neural Information Processing Systems**. Red Hook, NY, USA: Curran Associates Inc., 2020. (NIPS'20). ISBN 9781713829546. Disponível em: https://dl.acm.org/doi/abs/10.5555/3495724.3495883.

CAMBRIA, E.; WHITE, B. Jumping NLP Curves: A Review of Natural Language Processing Research [Review Article]. **IEEE Computational Intelligence Magazine**, IEEE, v. 9, n. 2, p. 48–57, maio 2014. ISSN 1556-6048. Disponível em: https://doi.org/10.1109/MCI.2014.2307227.

CHIU, N.-H.; HUANG, S.-J. The adjusted analogy-based software effort estimation based on similarity distances. **Journal of Systems and Software**, v. 80, n. 4, p. 628–640, abr. 2007. ISSN 0164-1212. Disponível em: https://www.sciencedirect.com/science/article/pii/S0164121206001701.

CHOETKIERTIKUL, M. *et al.* A Deep Learning Model for Estimating Story Points. **IEEE Transactions on Software Engineering**, v. 45, n. 7, p. 637–656, jul. 2019. ISSN 1939-3520. Disponível em: https://doi.org/10.1109/TSE.2018.2792473.

CHOI, S.; PARK, S.; SUGUMARAN, V. A rule-based approach for estimating software development cost using function point and goal and scenario based requirements. **Expert Systems with Applications**, Elsevier, v. 39, n. 1, p. 406–418, 2012.

CHOLLET, F. *et al.* **Keras**. 2015. https://keras.io.

COHN, M. **Agile Estimating and Planning**. USA: Prentice Hall PTR, 2005. ISBN 0131479415. Disponível em: https://dl.acm.org/doi/10.5555/1036751.

CONRADO, M. d. S. *et al.* A survey of automatic term extraction for Brazilian Portuguese. **Journal of the Brazilian Computer Society**, v. 20, n. 1, p. 12, maio 2014. ISSN 1678-4804. Disponível em: https://doi.org/10.1186/1678-4804-20-12.

CRYSTAL, D. **A Dictionary of Linguistics and Phonetics**. John Wiley & Sons, 2011. (The Language Library). ISBN 9781444356755. Disponível em: https://books.google.com.br/books?id=3ZPQVuSgDAkC.

DAWSON, C. W.; WILBY, R. An artificial neural network approach to rainfall-runoff modelling. **Hydrological Sciences Journal**, Taylor & Francis Group, v. 43, p. 47–66, 2009. ISSN 21503435. Disponível em: https://doi.org/10.1080/02626669809492102.

DENNY, M. J.; SPIRLING, A. Text Preprocessing For Unsupervised Learning: Why It Matters, When It Misleads, And What To Do About It. **Political Analysis**, Cambridge University Press, v. 26, n. 2, p. 168–189, abr. 2018. ISSN 1047-1987, 1476-4989. Disponível em: https://doi.org/10.1017/pan.2017.44.

DEVLIN, J. *et al.* BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. **CoRR**, abs/1810.04805, 2018. Disponível em: http://arxiv.org/abs/1810.04805.

FÁVERO, E. M. D. B.; CASANOVA, D.; PIMENTEL, A. R. SE^{3}M: A model for software effort estimation using pre-trained embedding models. **Information and Software Technology**, v. 147, p. 106886, 2022. ISSN 0950-5849. Disponível em: https://www.sciencedirect.com/science/article/pii/S0950584922000507.

FÁVERO, E. M. D. B. *et al.* Analogy-based Effort Estimation: A Systematic Mapping of Literature. **INFOCOMP Journal of Computer Science**, v. 17, n. 2, p. 07–22, 2018. Disponível em: https://infocomp.dcc.ufla.br/index.php/infocomp/article/view/565.

FLORIDI, L.; CHIRIATTI, M. GPT-3: Its Nature, Scope, Limits, and Consequences. **Minds and Machines**, v. 30, n. 4, p. 681–694, dez. 2020. ISSN 1572-8641. Disponível em: https://doi.org/10.1007/s11023-020-09548-1.

FONSECA, C. **Word Embedding: fazendo o computador entender o significado das palavras**. 2021. Disponível em: https://medium.com/turing-talks/word-embedding-fazendo-o-computador-entender-o-significado-das-palavras-92fe22745057.

GANEGEDARA, T. **Light on Math ML: Intuitive Guide to Understanding GloVe Embeddings**. 2021. Disponível em: https://towardsdatascience.com/light-on-math-ml-intuitive-guide-to-understanding-glove-embeddings-b13b4f19c010.

GARG, S.; SHARMA, R. K.; LIANG, Y. SimpleTran: Transferring Pre-Trained Sentence Embeddings for Low Resource Text Classification. **CoRR**, arXiv, abs/2004.05119, 2020. Disponível em: https://arxiv.org/abs/2004.05119.

GERS, F. A.; SCHMIDHUBER, J.; CUMMINS, F. Learning to forget: Continual prediction with lstm. **Neural Computation**, MIT Press Journals, v. 12, p. 2451–2471, 2000. ISSN 08997667. Disponível em: http://dx.doi.org/10.1162/089976600300015015.

GOLDBERG, Y. **Neural Network Methods in Natural Language Processing**. San Rafael, CA, USA.: Morgan & Claypool, 2017. (Synthesis Lectures on Human Language Technologies). ISBN 978-1-62705-298-6. Disponível em: https://doi.org/10.1007/978-3-031-02165-7.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. MIT Press, 2016. Disponível em: http://www.deeplearningbook.org.

GRAVES, A. Supervised sequence labelling. *In*: ____. **Supervised Sequence Labelling with Recurrent Neural Networks**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 5–13. ISBN 978-3-642-24797-2. Disponível em: https://doi.org/10.1007/978-3-642-24797-2_2.

GULLI, A.; KAPOOR, A.; PAL, S. **Deep Learning with TensorFlow 2 and Keras: Regression, ConvNets, GANs, RNNs, NLP, and More with TensorFlow 2 and the Keras API**. Packt Publishing, 2019. (Expert insight). ISBN 9781838823412. Disponível em: https://books.google.com.br/books?id=hebZzAEACAAJ.

HAENLEIN, M.; KAPLAN, A. A Brief History of Artificial Intelligence: On the Past, Present, and Future of Artificial Intelligence. **California Management Review**, v. 61, n. 4, p. 5–14, 2019. Disponível em: https://doi.org/10.1177/0008125619864925.

HAJ-YAHIA, Z.; SIEG, A.; DELERIS, L. A. Towards Unsupervised Text Classification Leveraging Experts and Word Embeddings. *In*: **Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics**. Florence, Italy: Association for Computational Linguistics, 2019. p. 371–379. Disponível em: https://aclanthology.org/P19-1036.

HAN, X. *et al.* Pre-trained models: Past, present and future. **AI Open**, v. 2, p. 225–250, 2021. ISSN 2666-6510. Disponível em: https://www.sciencedirect.com/science/article/pii/S2666651021000231.

HARRIS, C. R. *et al.* Array programming with numpy. **Nature**, Springer Science and Business Media LLC, v. 585, p. 357–362, 9 2020. Disponível em: https://doi.org/10.1038/s41586-020-2649-2.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. **The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition**. Springer New York, 2009. (Springer Series in Statistics). ISBN 9780387848587. Disponível em: https://doi.org/10.1007/978-0-387-84858-7.

HATCHER, W. G.; YU, W. A survey of deep learning: Platforms, applications and emerging research trends. **IEEE Access**, Institute of Electrical and Electronics Engineers Inc., v. 6, p. 24411–24432, 4 2018. ISSN 21693536.

HIRSCHBERG, J.; MANNING, C. D. Advances in natural language processing. **Science**, American Association for the Advancement of Science, v. 349, n. 6245, p. 261–266, jul. 2015. Disponível em: https://www.science.org/doi/full/10.1126/science.aaa8685.

HOCHREITER, S. Untersuchungen zu dynamischen neuronalen netzen. **Diploma, Technische Universität München**, v. 91, 1991.

HOCHREITER, S. *et al.* Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. A field guide to dynamical recurrent neural networks. IEEE Press In, 2001.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural Computation**, MIT Press Journals, v. 9, p. 1735–1780, 11 1997. ISSN 08997667.

HOUDT, G. V.; MOSQUERA, C.; NÁPOLES, G. A review on the long short-term memory model. **Artificial Intelligence Review**, Springer Science+Business Media B.V., v. 53, p. 5929–5955, 12 2020. ISSN 15737462. Disponível em: https://link.springer.com/article/10.1007/s10462-020-09838-1.

HOWARD, J.; RUDER, S. Universal language model fine-tuning for text classification. 1 2018. Disponível em: https://arxiv.org/abs/1801.06146.

HUNTER, J. D. Matplotlib: A 2d graphics environment. **Computing in Science & Engineering**, IEEE COMPUTER SOC, v. 9, p. 90–95, 2007.

HUSSAIN, I.; KOSSEIM, L.; ORMANDJIEVA, O. Approximation of COSMIC functional size to support early effort estimation in Agile. **Data & Knowledge Engineering**, v. 85, p. 2–14, 2013. ISSN 0169-023X. Disponível em: https://doi.org/10.1016/j.datak.2012.06.005.

IDRI, A.; AMAZAL, F. a.; ABRAN, A. Analogy-based software development effort estimation: A systematic mapping and review. **Information and Software Technology**, v. 58, p. 206–230, fev. 2015. ISSN 0950-5849. Disponível em: https://www.sciencedirect.com/science/article/pii/S0950584914001815.

IONESCU, V.-S. An approach to software development effort estimation using machine learning. *In*: **2017 13th IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)**. Cluj-Napoca, Romania: IEEE, 2017. p. 197–203. Disponível em: https://doi.org/10.1109/ICCP.2017.8117004.

IONESCU, V.-S.; DEMIAN, H.; CZIBULA, I.-G. Natural language processing and machine learning methods for software development effort estimation. **Studies in Informatics and Control**, v. 26, n. 2, p. 219–228, 2017. Disponível em: https://doi.org/10.24846/v26i2y201710.

JAIN, A. K.; MAO, J.; MOHIUDDIN, K. M. Artificial neural networks: A tutorial. **Computer**, v. 29, p. 31–44, 3 1996. ISSN 00189162.

JANCZAK, A. Identification of nonlinear systems using neural networks and polynomial models: A Block-Oriented Approach. **Lecture Notes in Control and Information Sciences**, Springer Berlin Heidelberg, Berlin, Heidelberg, v. 310, 2005. ISSN 01708643. Disponível em: http://link.springer.com/10.1007/b98334.

JORDAN, M. I.; MITCHELL, T. M. Machine learning: Trends, perspectives, and prospects. **Science**, v. 349, n. 6245, p. 255–260, 2015. Disponível em: https://www.science.org/doi/abs/10.1126/science.aaa8415.

JURAFSKY, D.; MARTIN, J. H. **Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition: United State**. Upper Saddle River, N.J: Pearson, 2000. ISBN 978-0-13-095069-7.

JURAFSKY, D.; MARTIN, J. H. **Speech and Language Processing (3rd ed. (draft))**. 2019. Disponível em: https://web.stanford.edu/~jurafsky/slp3/.

KANNAN, S. *et al.* Preprocessing techniques for text mining. **International Journal of Computer Science & Communication Networks**, v. 5, n. 1, p. 7–16, 2014.

KAO, A.; POTEET, S. **Natural Language Processing and Text Mining**. Springer London, 2007. ISBN 9781846287541. Disponível em: https://doi.org/10.1007/978-1-84628-754-1.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. 12 2014. Disponível em: https://arxiv.org/abs/1412.6980.

KOCAGUNELI, E. *et al.* Exploiting the essential assumptions of analogy-based effort estimation. **IEEE Transactions on Software Engineering**, IEEE, v. 38, n. 2, p. 425–438, 2011.

KOCAGUNELI, E. *et al.* Exploiting the essential assumptions of analogy-based effort estimation. **IEEE Transactions on Software Engineering**, v. 38, p. 425–438, 2012. ISSN 00985589.

KOHAVI, R. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. v. 14, mar. 2001.

LANE, D. M. **Online statistics education: A multimedia course of study**. Association for the Advancement of Computing in Education (AACE), 2003. Disponível em: http://onlinestatbook.com/.

LAURIOLA, I.; LAVELLI, A.; AIOLLI, F. An introduction to Deep Learning in Natural Language Processing: Models, techniques, and tools. **Neurocomputing**, v. 470, p. 443–456, jan. 2022. ISSN 0925-2312. Disponível em: https://www.sciencedirect.com/science/article/pii/S0925231221010997.

LEFFINGWELL, D. **Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise**. Pearson Education, 2010. (Agile Software Development Series). ISBN 978-0-321-68540-7. Disponível em: https://books.google.com.br/books?id=pTExbNmZwZUC.

LIU, Z.; LIN, Y.; SUN, M. Word Representation. *In*: LIU, Z.; LIN, Y.; SUN, M. (Ed.). **Representation Learning for Natural Language Processing**. Singapore: Springer, 2020. p. 13–41. ISBN 9789811555732. Disponível em: https://doi.org/10.1007/978-981-15-5573-2_2.

MAGUERESSE, A.; CARLES, V.; HEETDERKS, E. Low-resource languages: A review of past work and future challenges. 6 2020. Disponível em: https://arxiv.org/abs/2006.07264v1.

MANIKAVELAN, D.; PONNUSAMY, R. Software cost estimation by analogy using feed forward neural network. *In*: **Information Communication and Embedded Systems (ICICES), 2014 International Conference on**. Chennai, India: IEEE, 2014. p. 1–5. Disponível em: https://doi.org/10.1109/ICICES.2014.7033820.

MANNING, C. D.; RAGHAVAN, P.; SCHÜTZE, H. **Introduction to Information Retrieval**. 1. ed. USA: Cambridge University Press, 2008. ISBN 0521865719. Disponível em: https://dl.acm.org/doi/10.5555/1394399.

MCKINNEY, W. pandas: a foundational Python library for data analysis and statistics. **Python for high performance and scientific computing**, v. 14, n. 9, p. 1–9, 2011.

MENZIES, T. *et al.* Selecting best practices for effort estimation. **IEEE Transactions on Software Engineering**, IEEE, v. 32, n. 11, p. 883–895, 2006.

MHASKAR, H.; LIAO, Q.; POGGIO, T. When and why are deep networks better than shallow ones? **Proceedings of the AAAI Conference on Artificial Intelligence**, v. 31, 2 2017. Disponível em: https://ojs.aaai.org/index.php/AAAI/article/view/10913.

MIKOLOV, T. *et al.* Efficient Estimation of Word Representations in Vector Space. **arXiv:1301.3781 [cs]**, set. 2013. Disponível em: http://arxiv.org/abs/1301.3781.

MINSKY, M.; PAPERT, S. **Perceptrons.** Oxford, England: M.I.T. Press, 1969.

MOHARRERI, K. *et al.* Cost-Effective Supervised Learning Models for Software Effort Estimation in Agile Environments. **Proceedings - International Computer Software and Applications Conference**, IEEE Computer Society, v. 2, p. 135–140, ago. 2016. ISSN 07303157.

NASEEM, U. *et al.* A Comprehensive Survey on Word Representation Models: From Classical to State-of-the-Art Word Representation Language Models. **ACM Transactions on Asian and Low-Resource Language Information Processing**, v. 20, n. 5, p. 74:1–74:35, jun. 2021. ISSN 2375-4699. Disponível em: https://doi.org/10.1145/3434237.

NWANKPA, C. *et al.* Activation functions: Comparison of trends in practice and research for deep learning. 11 2018. Disponível em: https://arxiv.org/abs/1811.03378.

OCHODEK, M. Functional size approximation based on use-case names. **Information and Software Technology**, Elsevier, v. 80, p. 73–88, dez. 2016. ISSN 0950-5849.

O'KEEFFE, A.; MCCARTHY, M. **The Routledge Handbook of Corpus Linguistics**. 1. ed. London, UK: Routledge, 2010. ISBN 978-1-135-15363-2. Disponível em: https://doi.org/10.4324/9780203856949.

OPENAI. **Documentation - OpenAI API**. 2022? Disponível em: https://beta.openai.com/docs.

PEDREGOSA, F. *et al.* Scikit-learn: Machine Learning in Python. **Journal of Machine Learning Research**, v. 12, jan. 2012.

PENNINGTON, J.; SOCHER, R.; MANNING, C. D. GloVe: Global Vectors for Word Representation. *In*: **Empirical Methods in Natural Language Processing (EMNLP)**. Doha, Qatar: Association for Computational Linguistics, 2014. p. 1532–1543. Disponível em: http://www.aclweb.org/anthology/D14-1162.

PMI, P. M. I. **A Guide to the Project Management Body of Knowledge**. 4. ed. [*S.l.*]: Global Standard, 2008.

PMI, P. M. I. **A Guide to the Project Management Body of Knowledge and the Standard for Project Management**. 7. ed. Newtown Square, Pennsylvania: Project Management Institute, 2021. ISBN 978-1-62825-664-2.

RADFORD, A. *et al.* Improving language understanding by generative pre-training. 2018.

RADFORD, A. *et al.* Language models are unsupervised multitask learners. **OpenAI blog**, v. 1, n. 8, p. 9, 2019.

RASCHKA, S. **Python Machine Learning**. 1. ed. Packt Publishing, 2015. ISBN 9781783555147. Disponível em: https://books.google.com.br/books?id=GOVOCwAAQBAJ.

RASCHKA, S. *et al.* **Machine Learning with Pytorch and Scikit-Learn Develop Machine Learning and Deep Learning Models with Python.** 1. ed. Packt Publishing, 2022. ISBN 9781801819312. Disponível em: https://www.packtpub.com/product/machine-learning-with-pytorch-and-scikit-learn/9781801819312.

RASCHKA, S.; MIRJALILI, V.; SAFARI an O. M. C. **Python Machine Learning**. 3. ed. Packt Publishing, 2019. 336-336 p. ISBN 9781789955750. Disponível em: https://books.google.com.br/books?id=SYs-zQEACAAJ.

RIBEIRO, D. **Polissemia**. 2020. Disponível em: https://www.dicio.com.br/polissemia/.

ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. **Psychological Review**, v. 65, n. 6, p. 386–408, nov. 1958. ISSN 0033295X. Disponível em: https://psycnet.apa.org/journals/rev/65/6/386.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. **Nature 1986 323:6088**, Nature Publishing Group, v. 323, p. 533–536, 1986. ISSN 1476-4687. Disponível em: https://www.nature.com/articles/323533a0.

RUSSELL, S.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. Pearson, 2016. (Always learning). ISBN 9781292153964. Disponível em: https://books.google.com.br/books?id=XS9CjwEACAAJ.

SARRO, F.; PETROZZIELLO, A.; HARMAN, M. Multi-objective software effort estimation. *In*: **Proceedings of the 38th International Conference on Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2016. (ICSE '16), p. 619–630. ISBN 9781450339001. Disponível em: https://doi.org/10.1145/2884781.2884830.

SCHOFIELD, A.; MAGNUSSON, M.; MIMNO, D. Pulling Out the Stops: Rethinking Stopword Removal for Topic Models. *In*: **Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers**. Valencia, Spain: Association for Computational Linguistics, 2017. p. 432–436. Disponível em: https://aclanthology.org/E17-2069.

SHEPPERD, M. Software project economics: A roadmap. *In*: **2007 Future of Software Engineering**. IEEE Computer Society, 2007. p. 304–315. ISBN 0769528295. Disponível em: https://doi.org/10.1109/FOSE.2007.23.

SHIVHARE, J.; RATH, S. K. Software effort estimation using machine learning techniques. *In*: **Proceedings of the 7th India Software Engineering Conference**. New York, NY, USA: Association for Computing Machinery, 2014. (ISEC '14). ISBN 9781450327763. Disponível em: https://doi.org/10.1145/2590748.2590767.

SMITH, S. *et al.* Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, A Large-Scale Generative Language Model. **CoRR**, abs/2201.11990, 2022. Disponível em: https://arxiv.org/abs/2201.11990.

SOLAIMAN, I. *et al.* Release Strategies and the Social Impacts of Language Models. **CoRR**, abs/1908.09203, 2019. Disponível em: http://arxiv.org/abs/1908.09203.

SRIVASTAVA, A. *et al.* Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. 6 2022. Disponível em: http://arxiv.org/abs/2206.04615.

STERNBERG, R. J. Component processes in analogical reasoning. **Psychological review**, American Psychological Association, v. 84, n. 4, p. 353, 1977.

SUN, S. *et al.* Patient Knowledge Distillation for BERT Model Compression. **CoRR**, abs/1908.09355, 2019. Disponível em: http://arxiv.org/abs/1908.09355.

SZANDAŁA, T. Review and comparison of commonly used activation functions for deep neural networks. *In*: ____. **Bio-inspired Neurocomputing**. Singapore: Springer Singapore, 2021. p. 203–224. ISBN 978-981-15-5495-7. Disponível em: https://doi.org/10.1007/978-981-15-5495-7_11.

TOMAN, M.; TESAR, R.; JEZEK, K. Influence of word normalization on text classification. **Proceedings of InSciT**, v. 4, p. 354–358, 2006.

TRAPPENBERG, T. **Fundamentals of Machine Learning**. Oxford University Press, 2019. ISBN 978-0-19-882804-4. Disponível em: https://books.google.com.br/books?id=jsLADwAAQBAJ.

TRENDOWICZ, A.; JEFFERY, R. **Software Project Effort Estimation: Foundations and Best Practice Guidelines for Success**. Springer International Publishing, 2014. ISBN 9783319036298. Disponível em: https://books.google.com.br/books?id=iswkBAAAQBAJ.

VASWANI, A. *et al.* Attention Is All You Need. **Computing Research Repository**, abs/1706.03762, 2017. Disponível em: http://arxiv.org/abs/1706.03762.

WITTEN, I. H.; FRANK, E.; HALL, M. A. Chapter 5 - Credibility: Evaluating What's Been Learned. *In*: WITTEN, I. H.; FRANK, E.; HALL, M. A. (Ed.). **Data Mining: Practical Machine Learning**

Tools and Techniques (Third Edition). 3. ed. Boston: Morgan Kaufmann, 2011, (The Morgan Kaufmann Series in Data Management Systems). p. 147–187. ISBN 978-0-12-374856-0. Disponível em: https://www.sciencedirect.com/science/article/pii/B9780123748560000055.

WYNNE, M. (Ed.). **Developing Linguistic Corpora: A Guide to Good Practice**. Oxford, Oakville, CT: Oxbow Books, 2005. ISBN 978-1-84217-205-6.

YADAV, S. **Understanding Attention Mechanism | by Shashank Yadav | Medium**. 2019. Disponível em: https://shashank7-iitd.medium.com/understanding-attention-mechanism-35ff53fc328e.

YIN, X. *et al.* Fine-tuning and visualization of convolutional neural networks. *In*: **2017 12th IEEE Conference on Industrial Electronics and Applications (ICIEA)**. Siem Reap, Cambodia: IEEE, 2017. p. 1310–1315. ISBN 978-1-5090-6161-7. Disponível em: http://ieeexplore.ieee.org/document/8283041/.

YU, T.; ZHU, H. Hyper-parameter optimization: A review of algorithms and applications. arXiv, 3 2020. Disponível em: https://arxiv.org/abs/2003.05689.

ZHANG, C. *et al.* ESSE: An Early Software Size Estimation Method Based on Auto-Extracted Requirements Features. *In*: **Proceedings of the 8th Asia-Pacific Symposium on Internetware**. New York, NY, USA: Association for Computing Machinery, 2016. (Internetware '16), p. 112–115. ISBN 9781450348294. Disponível em: https://doi.org/10.1145/2993717.2993733.

ZHANG, D. *et al.* Chinese comments sentiment classification based on word2vec and SVMperf. **Expert Systems with Applications**, Pergamon, v. 42, n. 4, p. 1857–1863, 3 2015. ISSN 0957-4174. Disponível em: https://www.sciencedirect.com/science/article/pii/S0957417414005508.

ZHANG, S. *et al.* Opt: Open pre-trained transformer language models. arXiv, 5 2022. Disponível em: https://arxiv.org/abs/2205.01068v4.

ZHANG, W. *et al.* On definition of deep learning. *In*: **2018 World Automation Congress (WAC)**. Stevenson, WA, USA: IEEE, 2018. p. 1–5. ISBN 978-1-5323-7791-4. Disponível em: https://ieeexplore.ieee.org/document/8430387/.

ZHUANG, F. *et al.* A comprehensive survey on transfer learning. **Proceedings of the IEEE**, v. 109, p. 43–76, 1 2021. ISSN 0018-9219. Disponível em: https://ieeexplore.ieee.org/document/9134370/.

GLOSSÁRIO

embedding abreviação de *word embeddings*. 12, 14, 22–26, 44, 50, 52, 57, 58, 61–64, veja *word embedding*.

grid search método simples de otimização de hiperparâmetros, que realiza uma busca exaustiva em um espaço de busca. 6, 41, 53, 61, 63–65

k-fold método de validação cruzada, utilizado para a validação de modelos de aprendizado de máquina. 4, 39, 40, 50, 51, 53, 58, 61

one-hot forma de representação vetorial na qual uma posição do vetor é um bit alto (1) e todas as outras posições são bits baixos (0). 20, 21, 42

random search variação do algoritmo *grid search*, que escolhe aleatoriamente parte dos conjuntos de hiperparâmetros de busca. 41, veja *grid search*.

SkipGram técnica de aprendizado não-supervisionado utilizado para encontrar as palavras mais relacionadas à uma dada palavra. 23

stopword palavras comuns nos textos, que são irrelevantes na classificação de um documento. 19, 21

token instância ou sequência de caracteres de um documento, agrupados como uma unidade semântica a ser processada. 19, 27, 28, 36, 50, 56, 57, 62, 64

Transformer modelo de *deep learning* autosuficiente, que avalia suas entradas gerando uma representação de dados como saída. 25–28, 36, 47

word embedding termo usado para descrever a representação de palavras para análise de texto. 6, 12, 20, 22–24, 42–44, 47, 57, 58, 61, 63, 65

ANEXO A – Artigo - Revisão Sistemática da Literatura

Esse Anexo apresenta o artigo contendo a revisão sistemática da literatura, intitulado: *"Analogy based Effort Estimation: A Systematic Mapping of Literature"* (FÁVERO *et al.*, 2018). Esse artigo foi publicado em 2018 pelo INFOCOMP - Journal of Computer Science.

Analogy-based Effort Estimation: A Systematic Mapping of Literature

Article *in* Journal of Computer Science · December 2018

CITATION
1

READS
183

4 authors:

Eliane Maria de Bortoli Fávero
Federal University of Technology - Paraná/Brazil (UTFPR)
11 PUBLICATIONS **5** CITATIONS
SEE PROFILE

Dalcimar Casanova
Federal University of Technology - Paraná, Brazil
61 PUBLICATIONS **1,274** CITATIONS
SEE PROFILE

Andrey Ricardo Pimentel
Universidade Federal do Paraná
51 PUBLICATIONS **131** CITATIONS
SEE PROFILE

Roberto Pereira
Universidade Federal do Paraná
169 PUBLICATIONS **730** CITATIONS
SEE PROFILE

Analogy-based Effort Estimation: A Systematic Mapping of Literature

ELIANE MARIA DE BORTOLI FÁVERO[1]
ANDREY RICARDO PIMENTEL[2]
ROBERTO PEREIRA[2]
DALCIMAR CASANOVA[1]

Technological Federal University of Paraná - UTFPR
Academic Department of Informatics
Via do Conhecimento, Km 01, S/N
85503-390 - Pato Branco (PR) - Brazil[1]
Federal University of Paraná - UFPR
Department of Informatics Rua Evaristo F. F. da Costa, 418 - Jardim das Américas
80050-540 - Curitiba (PR) - Brazil[2]
[1] (elianedb,dalcimar)@utfpr.edu.br
[2] (andrey,rpereira)@inf.ufpr.br

Abstract. Many research initiatives have been developed in the field of Software Engineering, including the area of software estimation. Software effort estimation techniques based on analogy are applied from historical data of projects, obtained in the early stages of software development. In this context, this paper presents a systematic mapping of the literature, aim to elicit the state of the art on analogy-based software effort estimation techniques , indicating challenges and research opportunities. The mapping was done for the period from 2007 to 2017 and was conducted separately for each of the selected sources. The articles found were reviewed according to previously established research and selection criteria, according to the study objectives. Note that the model of estimation by analogy has received more attention and is presented as a promising and feasible technique in relation to the others. The techniques of Adaptive Neuro-fuzzy Inferences (ANFIS), Collaborative Filtering (FC), Radial Basis Functions (FBR) and Deep Learning present a gap to be explored. The results point to a demand for simple and practical estimation techniques, with emphasis on the estimation based on analogies and for the exploration of the artifacts generated in the initial phase of development, mainly in the textual format.

Keywords: systematic mapping; analogy; software effort estimation; analogy-based effort estimation.

(Received November 11th, 2018 / Accepted December 03th, 2018)

1 Introduction

Estimating software effort is a challenging and important activity in the software development process. This activity depends on the success of other crucial aspects of a project that directly impact the quality of the software product developed, predominantly the time under which the software was developed and the budget constraints. The success of any particular software project depends greatly on the accuracy of its effort estimates [39]. An accurate estimate assists in contract negotiations, scheduling and synchronization of project activities and efficient allocation of resources. [75], shows an increase in the rate of software design flaws –especially in the year 2012 –which resulted in budget and / or schedule overflows.

There are currently different types of development

processes that have differing impacts on planning and estimating software projects. Ways to generate effort estimates to achieve better results have been one of the focuses of the Software Engineering industry for quite some time [31, 11, 41, 7, 10, 4]. These experiments have continuously explored computational techniques individually or in combination, seeking to achieve better levels of precision in effort estimation.

Several classifications for software effort estimation models have been proposed in the last decade and include small differences according to the authors' point of view. For instance, [69] classifies software effort estimation in algorithmic/parametric and non-algorithmic terms. The former includes those that use mathematical or algorithmic models (applied to project attributes) to calculate their estimate, e. g. COCOMO II [15] and Function Point Analysis [24]. The latter relies on machine learning techniques, and uses historical project data to generate learning models to predict future estimates [49], for example, the use of regression models [2, 81] e classification algorithms [46, 59].

Shepperd et al.[67], in contrast, classifies effort estimates in software projects into three categories: 1. Human-centric (e.g. expert judgment); 2. model-based (e.g. COCOMO); and 3. induced prediction techniques [53, 2]. Although they are widely used - especially in agile models, techniques relying on human determination problems because they become very subjective, often generating distorted estimates with excess of optimism, for example. Model-based techniques, by contrast, use replicable methods to produce estimates and can be more objective than those with [45]. Among the many induced predicted techniques, the main types used are linear regression, neural networks, and analogy [46, 59]. In order to use these techniques, it is interesting that the training data available are independent of algorithmic/parametric models.

Chiu et al.[21] highlight a fourth model: the analogy-based effort estimation (ABEE). This method has been widely used and aims to identify similarity between projects already carried out, thus generating the most approximate estimate for a new project. The origin of this method can be attributed to a study performed by [72], which identifies it as a viable approach to predicting estimates. Studies involving ABEE are commonly associated with Artificial Intelligence techniques (also associated with historical project data).

According to [37], the accuracy of the estimate is improved when the analogy is combined with another technique to generate estimates. Artificial Intelligence techniques are useful regardless of the mode used to approximate similar data. Fuzzy systems, genetic algorithms, case-based reasoning, and collaborative filtering are techniques that improve ABEE performance.

Approaches based on analogy have shown promise in the field of software effort estimation, and its use has increased among researchers in this area [39]. Authors, such as [37], classify analogy-based technique as a machine learning technique. This technique has been advocated as a potential method for efficient effort estimation, since it allows modeling the complexity between the effort and the variables included in the context of the software project (e.g. team data, project data), elements which have a relationship that is normally not linear. Wen et al.[78] carried out a systematic review of the literature in which they identified eight types of machine learning techniques. The Case-based reasoning (CBR) and artificial neural networks (ANN) were the most used techniques for estimating effort, representing 37% and 26%, respectively.

Idri et al. [37] in turn performed a systematic literature review on ABEE and found out that these techniques outperform other prediction techniques. Some advantages of this estimation technique as highlighted by the authors include: 1) they present a tendency to produce acceptable estimates, surpassing other estimation techniques (e. g. human-centered models and model-based estimates); 2) they generate models that relate the effort and attributes of the project context, despite the inherent complexity, resulting in reproducible models to produce estimates [46] (e. g. linear regression, neural networks and analogies [68, 50] and 3) unlike prediction techniques such as Artificial Neural Networks –such as black boxes –ABEE techniques are similar to human reasoning, which makes use of analogies in previous experiences in facilitating effort estimation.

However, ABEE techniques are still limited because they can not adequately handle categorical attributes measured on a nominal or ordinal scale, such as the complexity level of a requirement and/or the area of a project [39]. For this reason, the study considers the application of complementary techniques, such as machine learning.

ABEE is fit for both agile and traditional models as long as the estimation approach is based on previous team experiences to estimate software projects. A more latent disadvantage in agile models is that the estimation process is usually conducted in discussion among team members, as there are no formal conventions for the use of historical data. Another disadvantage of both development models is that there could be some inattention to exact detail in the process, these data may not have been properly recorded and could generate improper results.

Another challenge (especially in agile models) is the scarcity of data and project requirements available in the early stages of the development process. The basic requirement specification used in these models is the user history (or user requirements), and is generally written without formal convention [35]. A user story is a brief specification of user needs [25]. Informality is related to the lack of standardization in the specification of user stories, which are usually presented as unstructured text, which generates the need for adequate text exploration techniques in order to generate good characteristics to be used by techniques of Artificial Intelligence.

This paper presents a systematic mapping of the literature on the application of ABEE techniques in different contexts, looking at the discovery of research opportunities in the estimation of software effort. This systematic mapping is a necessary step to elicit the state of the art in ABEE, considering that the studies in this category of estimation have increased considerably in recent years.

The main goals of this paper are: 1. to provide a mapping of the studies in ABEE regarding: publication sources, research approach, types of contribution, techniques used, etc. and 2. to identify the resources used as inputs to the ABEE estimation process in order to identify other possible research gaps.

The next section presents the study plan developed to conduct this systematic mapping, followed by the presentation of the answers to the research questions and the discussions about the results and possible areas for ABEE research. Finally, the conclusion about this mapping is presented.

2 Methodology

The systematic mapping process followed the guidelines of [58] for drawing up the study. This plan specifies search expressions, search strategies, search engines, inclusion/exclusion criteria of studies, and data systematization, analysis and synthesis.

Systematic mapping studies are a type of systematic literature review that aims to collect and organize research articles related to a specific topic [58, 3, 42]. This type of study requires a careful and detailed search process, with well-defined inclusion and exclusion criteria [18]. According to [58], a systematic mapping usually presents broader research questions than a systematic review, primarily concerned with the structuring of a research area.

2.1 Research questions

Goal: this systematic mapping aims to identify studies related to ABEE techniques, being designed to provide answers to the following research questions:

- Q1.What is the distribution of ABEE publications over time?
  Justification: to verify how ABEE in research has evolved in the last years.
- Q2.What are the most common publication sources for ABEE publications?
  Justification: identify sources of publication that are more relevant to the topic.
- Q3.What are the countries of origin of ABEE publications and its main authors?
  Justification: to monitor the research carried out in the area and enable the exchange of experiences.
- Q4.What types of contributions do the ABEE studies present?
  Justification: to verify the volume of practical and theoretical work carried out in ABEE.
- Q5.What databases are used in the selected studies?
  Justification: to know the characteristics of the databases used in the studies carried out
- Q6.What solutions (methods, techniques, models) have been proposed in ABEE?
  Justification: to map the diversity of ABEE solutions available in the literature, independent of the development process adopted, and classify them to identify trends and / or common aspects within this area.
- Q7.What resources are used as input to the mapped estimation techniques in order to generate ABEE?
  Justification: to identify resources used as inputs to prediction techniques already studied.

2.2 Search strategy

The search was planned to cover the largest number of studies about software effort estimation, without specifying, at that moment, the estimation and development process model. Thus, the search expressions presented in Table 1, for each search engine, were selected and calibrated. These search engines were used because they appeared as the main sources of search and as the most used digital libraries for systematic studies [17, 26, 37].

Table 1: Search expressions

IEEE Explorer	**Number of articles returned**
((âestimating softwareâ OR âestimation softwareâ OR âeffort estimationâ OR âestimation of softwareâ OR âsoftware effortâ) AND ("Abstract":âestimation softwareâ OR "Abstract":"estimating softwareâ OR "Abstract":"effort estimationâ) AND ("Document Title":âestimating softwareâ OR "Document Title":âeffort estimationâ OR "Document Title":âsoftware effortâ OR "Document Title": âestimation softwareâ OR "Document Title":âestimation of softwareâ))	580
ACM Digital Library	
(+estimating +software +estimation +software +effort +estimation +estimation +of +software +software +effort) AND acmdlTitle:(+estimating +software +effort +estimation +software +effort +estimation +software +estimation +of +software)	91
Science Direct	
- With at least one of the words in the title: "estimating software" OR "estimation software" OR "estimation of software" OR "effort estimation" OR "estimating software" OR "software effort"	109
Springer	
- With at least one of the words in the title: "estimating software" OR "estimation software" OR "estimation of software" OR "effort estimation" OR "estimating software" OR "software effort"	174

The research considered papers published from 2007 to the 2017, including the first quarter of 2018, and was conducted separately for each search source. Other relevant search sources such as Google Scholar and DBLP were not used because their results were included in the other sources consulted.

2.3 Inclusion/Exclusion Criteria

The selection of the studies occurred from the reading of the title, summary and keywords of the resulting publications. Two filters were applied on the 954 studies retrieved by the search expressions (see Table **??**) following inclusion criteria were applied in the 1st filter (CI1F):

- CI_1F–1: The study defines and/or presents theoretical and practical aspects aimed at the recovery/ generation of effort estimates in software projects.
- CI_1F–2: The study investigates, compares or evaluates proposed methods for the recovery/ generation of effort estimates in software projects.
- CI_1F–3: The study analyzes the application of methods aimed at the recovery/generation of effort estimates in software projects.

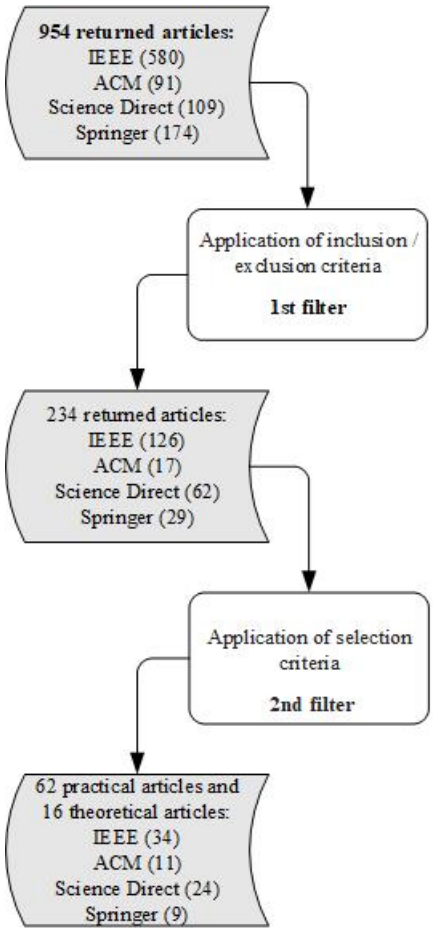


Figure 1: Selection process of publications

The following exclusion criteria for the first filter (CE_1F) were applied excluding papers that:

- CE_1F–1: Does not address an estimate of effort or time in software development projects.

- CE_1F–2: Uses only parametric or algorithmic models.
- CE_1F–3: Presents only future works.
- CE_1F–4: Are duplicates of a study already selected.
- CE_1F–5: Do not present any type of investigation, comparison, evaluation or application of methods aimed for ABEE in software projects.

When there were duplicates of articles referring to the same study, the most recent one was considered. For the first filter, the title, abstract and keywords were read, if necessary, the whole text was analyzed. For the remaining 234 articles inclusion criterion of the second filter (CI2F), considered the entire publication (full article and summary article published in conferences or journals), should have contained at least one of the following elements for the recovery/generation of effort estimates in software projects:

- CI_2F–1: are practical studies in ABEE;
- CI_2F–2: General (includes surveys, reviews and systematic mapping in ABEE).

When applying the criteria of the second filter, 78 relevant articles were retrieved, distributed among the selection criteria presented, of which 62 are classified as practical studies, which will be explored in the results of this mapping. The Figure 1 shows the search methodology used in this mapping, as well as the number of studies returned for each search in their respective digital library and the amounts of studies remaining after the application of the first and second filters.

2.4 Extraction of data

To answer the research questions of this systematic mapping was defined a set of data to be extracted from the selected articles was defined. Table 2 presents the data extracted from the selected articles.

Table 2: Data to be extracted from publications

Attributes	**Research questions**
Article title	Q1
Year of publication	Q1
Source of publication	Q2
Name (s) of the author(s)	Q3
Country of publication	Q3
Type of contribution (theoretical, systematic review of literature, systematic mapping, Surveys, practice - tool, model, method, technique, comparison)	Q4
Database used in the search: name, public/ private	Q5
IA techniques applied to the estimation: Fuzzy Systems, Fuzzy Analogy, Genetic Algorithms, Artificial Neural Networks, Statistical Models, Decision Trees, Naive Bayes, Case-based Reasoning (CBR), Regression Models â CART, MLR, SWR, Collaborative Filtering, Bees Algorithm, Similarity Measures , Support Vector Machines, Radial Base Function, Differential Evolution, Classical Analogy, Bayesian Regression	Q6
By Analogy (Yes/No): Consider the existence of historical data with or without the use similarity measures	Q6
In agile process model: Yes / No	Q6
Resources used as input in estimation techniques: numerical, textual, mixed	Q7

2.5 Threats to Validity

This systematic mapping draws on a protocol theoretically advocated by [58], enabling a correct and consistent research process, guaranteeing aspects such as generability and descriptive validity. Moreover, it is important to cite potential threats to the validity of the research: i) the use of search expressions and ii) the theoretical validity when evaluating inclusion and exclusion criteria and data extraction.

Regarding the search expressions it should be noted that it was not possible to use the same search expression on all search engines due to differences in the input format for this expression, which may not guarantee complete coverage of all related, relevant studies that were returned. Regarding the theoretical validity, different interpretations for the inclusion and exclusion criteria may occur, as well as for the data to be extracted from the studies, which depends on the researcher's bias, which is common when this analysis is done individually by the researchers.

3 Mapping Results

This section presents the results related to the proposed systematic mapping, according to the research questions previously defined. Throughout the presentation of the results, the necessary discussions are development, highlighting data and techniques used and pointing out to research gaps.

Regarding the distribution of ABEE publications over time (Q1), Figure 2 shows the distribution of selected publications over the last ten years. It can be observed that there has been a growth in research in the

field of software effort estimation, mainly in ABEE, the focus of this mapping, especially perceived from 2014, and rising in the following years, especially in 2016. This can be partially explained by the increasing need to improve software development processes and, because of this, the importance of an estimate of quality.

According to [29], the model of analogy-based estimation has received more attention and is presented as a promising and viable technique compared to other methods. One of the latent research aspects of analogy-based effort estimation is how to predict the effort of software projects when they are described by mixed numeric and categorical data [4]. In addition, [38] cites that models by analogy can be easily understood by users, as they resemble the human approach to problem-solving, unlike "black-box" models such as artificial neural networks.

The Table 3 presents the most common publication sources for ABEE studies (Q2), showing that 22% of the selected publications were published in a small set of journals and another 5% on specific conferences/symposium, with the most outstanding journals being the Journal of Systems and Software (JSS), Information and Software Technology (IST), Institution of Engineering and Technology (IET) and Applied Soft Computing. Another 73% was published in various newspapers and conferences, with only few occurrence in each.

Among the countries with the highest number of publications in ABEE, (Q3) are Marocos (12), Canada (10), India (8), China (6) and Iran (6), Jordan and USA appear with 5, United Arab Emirates e Japan appear with 3, Malaysia, Thailand and United Kin appear with 2 publications.

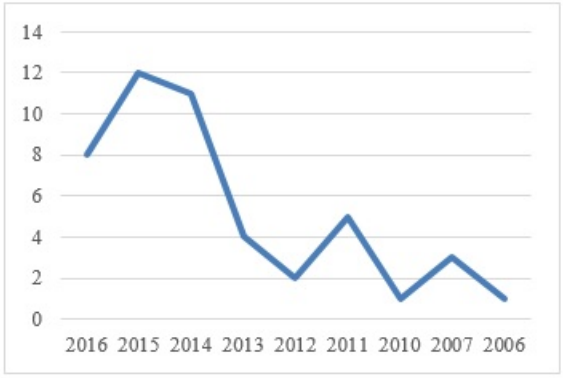


Figure 2: Number of publications by year after 2nd filter

For responding to Q4, which refers to the types of contributions presented by the studies in ABEE, it is re-

Table 3: Publication sources vs. occurrence

Title	Type	Number	%
Journal of Systems and Software (JSS)	Journal	6	10
Institution of Engineering and Technology (IET)	Journal	4	6
Applied Soft Computing	Journal	3	5
Symposium On Applied Computing (SAC)	Symposium	3	5
Other	Journals and Conferences	46	74

sults show that: 79% (62 publications) of them are practical contributions, especially those that are classified as methods/techniques, be they innovations or improvements of existing methods/techniques. Only 21% (16 publications) of the contributions are theoretical (systematic reviews of the literature, systematic mapping, surveys).

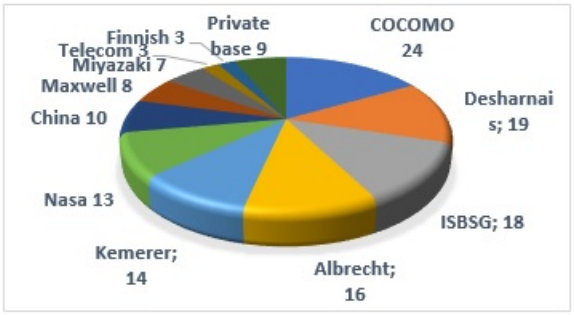


Figure 3: Databases used in surveys

In order to answer Q5, about the databases used in the studies, it was observed among the practical selected studies that approximately 94% (135) of them Figure 3 used historical data from public databases available for research on the development of software projects (e.g. ISBSG, Desharnais, Albrecht). Only 6% (9) of the studies were carried out using private databases obtained from software development organizations. The studies, carried out specifically with agile effort estimation techniques, were based on data provided by industry engineers and other academic studies performed. The study of [**?**] points out that their study was the first time a database was used and was made available for future studies involving requirements in agile models. As specified in Table 2 and Figure 4, the most used techniques in ABEE are Statistical Mod-

els (STM) and Fuzzy Systems (FS), followed by Classical Analogy (CA), Regression Models (RM), Case Based Reasoning (CBR) and Artificial Neural Network (ANN).

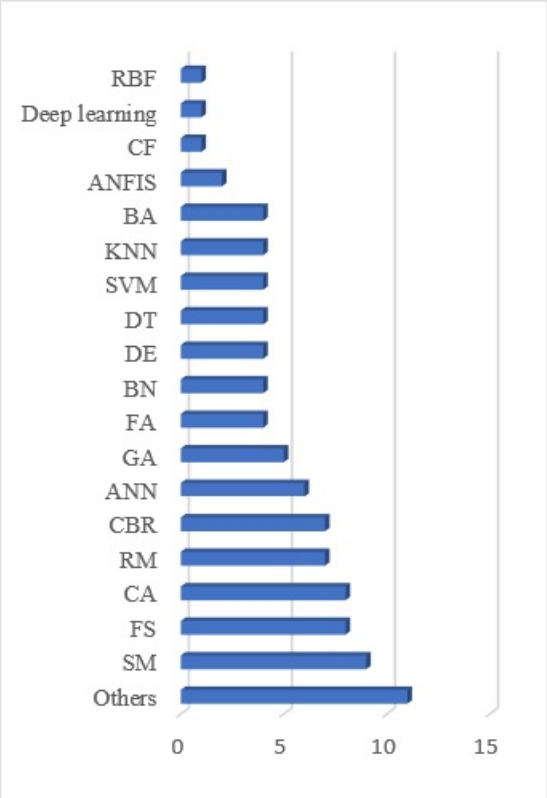


Figure 4: Most used techniques

The Figure 4 relates to the content to Q6, about the solutions presented by the studies analyzed. Upon review of the studies presented, which included the use of Adaptive Neuro-fuzzy Inferences (ANFIS), Collaborative Filtering (FC), Radial Basis Functions (FBR) and Deep Learning, it is possible to identify gaps in research particularly the exploitation of textual artifacts. The study by [22, 23], which applied deep learning to explore the texts of requirements in order to obtain an estimate of the effort of software projects. Other techniques that only have one occurrence and have potential for further research (included in âOtherâ) in which [62] analyzed the Firefly Algorithm and [52] which analyzed the Satin Bower Bird Optimization Algorithm. These three studies present a gap in research, especially in relation to the exploration of textual exploitation.

Considering Q7, only 11% (7 studies) are applied to the agile models of development Table 4, all of which are practical studies, including theoretical and practical studies on estimation by analogy.

The Table 3 presents the practical studies developed in the context of software estimation by analogy and that make use of textual requirements in the initial phase. Each study listed has the techniques used, the development model and its objective.

The Q8 investigates the type of data used as input in the estimation techniques studied Figure 5. Only 13% (8) of the practical studies analyzed use the text exploration of artifacts generated in the initial stages of development (e.g. user stories or other requirements documents) as input to the ABEE. From these, 50% (4) apply to the context of agile models and another 50% to traditional models of development.

The other 87% (54) of the studies presented in this table use other basic design attributes and software size metrics (e.g. points per function, points per story, COSMIC functional size).

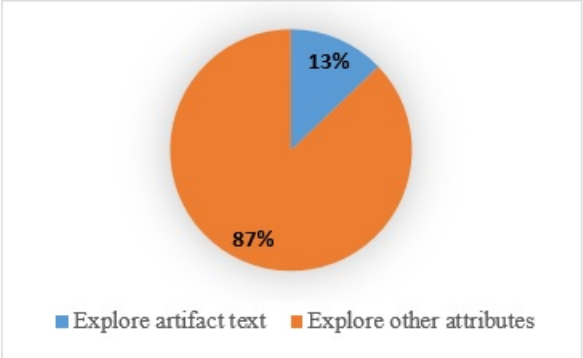


Figure 5: Input data for the estimation process

Table 4: Studies in effort estimation by analogy/development process model

	Number of studies	%	Identification of the publication
Agile models	7	11	[22, 30, 51, 2, 57, 27, 40]
Others	55	89	[45, 69, 20, 21, 10, 11, 9, 37, 82, 38, 39, 12, 73, 61, 29, 4, 49, 48, 9, 28, 43, 46, 47, 7, 19, 55, 81, 56, 6, 66, 32, 1, 63, 52, 5, 8, 14, 80, 79, 62, 59, 70, 77, 54, 38, 71, 13, 65, 34, 60, 36, 33, 64, 74]

Table 5: Studies on software effort estimation using textual requirements

Publication ID	Techniques used	Exploited data	Development model	Overview
[51]	ME, AM (various algorithms)	user history text + history attributes (id, project, estimate)	agile	An automated estimation methodology called "Self-Estimating" was proposed, complementing the Agile Planning Poker model. The Self-Estimating Leverage features extracted from user stories and their actual effort time. The approach is justified by evaluating alternative machine learning algorithms for prediction. It was shown that the selected machine learning methods performed better than Planning Poker's estimates in later phases of a project. This estimation approach is evaluated for accuracy, applicability and value, and results are presented in real-world environment.
[2]	linear regression, SVM, RBF	text of user stories	agile	A method was proposed to predict the effort based on user stories produced from agile models. The proposed method is based on the extraction of predictors from user histories and was applied to two agile software projects in the industrial context. It has been shown that this effort estimate works reasonably well if the user stories are written in a structured way.
[35]	Stanford Parser [44] (tagger POS of Brill [16] and a morphological stemmer), classifiers Naive Bayes, and a logistic classifier	requirement text	traditional	This work aims to develop a tool that automatically realizes a faster approach of the COSMIC size without requiring the formalization of the requirements, demonstrating its applicability in popular agile processes.
[22]	Deep learning: Long Short Term Memory (LSTM) and Recurrent Highway Networks (RHN)	user story text	agile	The model consists of a combination of two powerful deep learning architectures: long-term memory (LSTM) and recurrent road network (RHN). The forecasting system is trainable from start to finish with raw input data for forecast results without any manual resource engineering. The model learns from the point estimates of the team's previous story to predict the size of new stories. It is used in conjunction with (rather than a substitute for) existing estimation techniques practiced by the team.
[81]	Semantic analysis of block and word-level requirements and regression algorithms	requirement text	traditional	An initial software size estimation method has been proposed which can extract semantic features from natural language requirements automatically and construct size estimation models for a project by analogy.

[40]	Artificial neural networks	requirement text	agile	The proposed method uses techniques, such as word merges, to create a system that can estimate effort using only basic project management metrics and textual task descriptions. An artificial neural network was used to automate the task of estimating effort. The method was evaluated with real data from industrial software projects. The results outweigh some of the related libraries.
[56]	Decision tree C 4.5, bayesian networks	requirement text	traditional	From a UML use case diagram or a list of use-case names, automate the COSMIC and IFPUG FPA functional size approximation. The proposed method consists of a two-step process to approximate the functional size of applications based on use-case objectives. First, the names of the use cases, expressed in natural language, were assigned to each of the thirteen categories. In the second step, information on use-case objective categories and historical data was used to construct prediction models and use them to approximate size in COSMIC and Function Points.
[6]	multiple regression analysis	requirement text	traditional	To streamline measurement of size and effort estimation, this study explores the correlations between measures in the problem domain, such as the number of distinct nouns and distinct verbs in the requirement artifacts, and the solution domain measures, such as the number of classes and methods in the corresponding object-oriented software. Twelve commercial software projects were analyzed and multiple analysis were implemented to develop an estimation model for solution domain metrics in terms of problematic domain metrics. The results suggest that for the projects examined, it is possible to use problematic domain measures to make plausible predictions for solution domain metrics.

All studies presented in Table 5 make use of textual requirements of the initial phases of the development process, of which, three specifically explore user histories [51, 2, 22]. Moreover, five of these studies explore the text of other initial requirements [35, 81, 40, 56, 6].

Of the studies presented in Table 3, it is interesting to note that only one of them integrates data extracted from initial textual requirements with attributes of user history (e.g. complexity, priority) and none of the studies integrates data extracted from the text of the initial requirements with attributes of the project and development team.

4 Challenges and Opportunities

The systematic mapping has identified other studies (research, mapping or systematic review of literature), but none of them is a specific review of studies presenting techniques in ABEE.

Regardless of the development model adopted, the existing techniques vary greatly at their precision level, with little consensus in relation to different development contexts. For [76], more studies are needed on estimates in the context of the development of agile software that, also takes into account other predictors of effort, which can be checked as the results of the article.

Considering software development in general, initial requirements are generally textual and, therefore appear as a potential resource to be explored to obtain estimates based on a model by analogy. In agile processes (such as Scrum) –as in many traditional processes –the estimation is usually completed by human participation and is usually based on the experience and historical data of past projects; these projects generally focus on the development effort of each functionality without considering the context in which these projects were conducted.

It should be considered that human factors can also affect the development of projects, because they involve subjective aspects (e. g. the turnover of team members, the lack of experience of one or more team members in certain types of projects or technologies, etc). Such aspects are difficult to measure, and increase the likelihood of possible estimation errors. Therefore, both project-related factors and human factors should be considered when making estimations. However, the systematic mapping study revealed an absence of both factors in the estimation process.

When analyzing the databases used in the estimation studies Figure 3, it was observed that most of them store data from the project context, with little emphasis on the human data involved in development. In addition, only one study used a requirements-based database, which is geared to agile models and is in English. It would be invaluable if this study was made available in other languages and accompanied by the consideration of other facts besides the development effort.

From the point of view of the artifacts used in the estimation process, a considerable portion (50%, 4 studies) of studies within the agile context and others portion of 50% in the traditional context aim at the exploration of textual artifacts to estimate development effort. These studies used the text of requirements and their estimation metric, both individual and used in combinationâas attributes in the initial requirements. Both variants (individual or in combination) neglected to consider data extracted from the text of user stories to the context of the project (especially human factors) –fundamental in estimating effort.

Most of the studies presented use Artificial Intelligence techniques based on learning in the estimation of software effort. These techniques present limitations in the use of variables, either in terms of quantity or its format (numerical or textual). For this reason, models for automatic generation of estimates fail to consider some variables or do not use textual variables; some even still are semi-automatic models and employ the experience of the project manager and the team to insert these variablesâa subjective process with room for error. Thus, an opportunity would be to integrate AI techniques appropriately in order to include all the variables involved in the estimation process. Retrieving the most approximate estimate from historical data of projects (ABEE) presents itself as a promising field of research, always aiming at better results.

ABEE methods require human intervention, either in the recording of previous project data, in the analysis and implementation of the requirements, in the validation of the results obtained from the estimation, among other activities.

Based on the results from the mapping, it is possible to identify critical aspects involved in ABBE. Fig 6 draws on the Organizational Onion to show that these aspects are related to different aspects of formality in an organization, where technical aspects are part of formal aspects that exist in the context of informal activities and practices.

As Figure 6 suggests, informality seems to play an important role in software estimation. Activities carried out in a random manner, that is, there is not necessarily a pre-established pattern for its realization, not for its registration. For example, when done manually, estimates depend on discussions among team members about their experience in similar projects to arrive at

Figure 6: Informal, formal and technical levels of the agile estimation process

the most accurate estimate possible, and for this, historical data are not always available, nor are ways to measure the effort destined to these meetings to estimate a project. As previously mentioned, involving human aspects makes estimation somewhat subjective, since it involves different experiences, expectations, motivations, points of view, etc.; for this reason, the importance of the formalization of the activities carried out at the informal level is important, and detailed more specifically in Figure 6. Especially in agile process models, the role of personal decision making is an imperative because it ensures the dynamics and agility intrinsic to the development process. In this process model, the estimation of effort for the realization of a functionality usually occurs through the use of algorithmic models (e.g. estimation of points by history or points by use cases), which are assigned by the members of the development team [57]. When historical data is available, estimate is done in an analogous way, often manually or semi-automatically, which does not rule out the human role in this process, at least to evaluate the resulted estimate.

Therefore, an important contribution would be the formalization of the development process, as shown in the Formal level of Figure 6, which would allow all these aspects to be considered in the generation of the estimates, thus reducing the human effort in the estimation process, resulting in more accurate estimates generated from actual data based on the context of each project (e.g. type of project, complexity, level of developer experience, technology, among others).

On the hand, involving human aspects makes estimation somewhat subjective, since it involves different experiences, expectations, motivations, points of view, among others. On the other hand, it is a rich source of information that carries relevant knowledge from the context where the software development takes place, including people, the environment, organizational structure, stakeholders and so on. Therefore, the formalization of the activities carried out at the informal level is important to help taking advantage from this informal richness. Especially in agile process models, the role of people in decision making becomes critical in order to ensure the dynamics and agility intrinsic to the process.

In agile models, requirements are defined in a rather lean way, with minimal bureaucracy. In the case of generating a feasible estimate from data extracted from system's requirements (e.g. from user stories texts), this specification of requirements requires a standard structure. According to [22], although some initiatives can be found, this standard structure is still non-existent or not widley accepted in agile models, suggesting the need for research to reduce this gap.

Such an structure may make it to formalize requirements in order to ensure that they present the information necessary to generate a consistent estimate, preserving the richness of the informal context without overloading or restricting the activity.

Approaches for the automatic or semi-automatic generation for agile estimation, based on artifacts created in the initial phase of development, are still presented as a challenge. It is important to be aware of not losing the essence and principles of agile models, taking advantage of their flexibility and paving room for formal and automatic approaches.

5 Conclusion

This paper presented the main results from a systematic mapping of Analogy-based software effort estimation studies. The mapping covered studies published from 2007 to 2017 (including the first quarter of 2018), indexed by the main digital libraries (IEEE-Explorer, ACM Digital Library, Springer, Science Direct) available for Computer Science research.

Based on the results of the mapping, it was possible to observe that the model of estimation by analogy has received more attention and is presented as a promising and feasible technique in relation to other methods. One possible reason for that is its similarity with techniques that use classic analogies. The techniques of Adaptive Neuro-fuzzy Inferences (ANFIS), Collaborative Filtering (FC), Radial Basis Functions (FBR) and Deep Learning present a gap to be explored, as they have still been little explored for estimation purposes.

In agile models, the estimate is usually generated based on the experience of the people involved in the development process, even if there is a development history. Typically, the development history stores only data from the individual project itself, with little or no data on human aspects involved. This can be explained because the estimation process in agile environments is still essentially manual, with direct human intervention. In addi-

tion, there are limitations in existing project management tools for the effective exploitation of this historical data. Therefore, there is a demand for research on the software effort estimation process by analogy in the agile context allied to the use of data extracted from real agile contexts.

6 Bibliograph References

References

[1] Abnane, I., Idri, A., and Abran, A. Empirical evaluation of fuzzy analogy for software development effort estimation. In *Proceedings of the Symposium on Applied Computing*, pages 1302–1304. ACM, 2017.

[2] Abrahamsson, P., Fronza, I., Moser, R., Vlasenko, J., and Pedrycz, W. Predicting development effort from user stories. In *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on*, pages 400–403. IEEE, 2011.

[3] Acuña, S. T., Castro, J. W., Dieste, O., and Juristo, N. A systematic mapping study on the open source software development process. In *Evaluation & Assessment in Software Engineering (EASE 2012), 16th International Conference on*, pages 42–46. IET, 2012.

[4] Amazal, F. A., Idri, A., and Abran, A. Improving fuzzy analogy based software development effort estimation. In *Software Engineering Conference (APSEC), 2014 21st Asia-Pacific*, volume 1, pages 247–254. IEEE, 2014.

[5] Araújo, R. d. A., Oliveira, A. L., and Meira, S. A class of hybrid multilayer perceptrons for software development effort estimation problems. *Expert Systems with Applications*, 90:1–12, 2017.

[6] Ayyildiz, T. E. and Koçyigit, A. A case study on the utilization of problem and solution domain measures for software size estimation. In *Software Engineering and Advanced Applications (SEAA), 2016 42th Euromicro Conference on*, pages 108–111. IEEE, 2016.

[7] Azzeh, M. Adjusted case-based software effort estimation using bees optimization algorithm. *Knowlege-Based and Intelligent Information and Engineering Systems*, pages 315–324, 2011.

[8] Azzeh, M. and Nassif, A. B. A hybrid model for estimating software project effort from use case points. *Applied Soft Computing*, 49:981–989, 2016.

[9] Azzeh, M., Nassif, A. B., and Minku, L. L. An empirical evaluation of ensemble adjustment methods for analogy-based effort estimation. *Journal of Systems and Software*, 103:36–52, 2015.

[10] Azzeh, M., Neagu, D., and Cowling, P. I. Analogy-based software effort estimation using fuzzy numbers. *Journal of Systems and Software*, 84(2):270–284, 2011.

[11] Bardsiri, V. K., Jawawi, D. N. A., Bardsiri, A. K., and Khatibi, E. Lmes: A localized multi-estimator model to estimate software development effort. *Engineering Applications of Artificial Intelligence*, 26(10):2624–2640, 2013.

[12] Bardsiri, V. K., Jawawi, D. N. A., Hashim, S. Z. M., and Khatibi, E. Increasing the accuracy of software development effort estimation using projects clustering. *IET software*, 6(6):461–473, 2012.

[13] Benala, T. R. and Bandarupalli, R. Least square support vector machine in analogy-based software development effort estimation. In *Recent Advances and Innovations in Engineering (ICRAIE), 2016 International Conference on*, pages 1–6. IEEE, 2016.

[14] Benala, T. R. and Mall, R. Dabe: Differential evolution in analogy-based software development effort estimation. *Swarm and Evolutionary Computation*, 38:158–172, 2018.

[15] Boehm, B., Abts, C., and Chulani, S. Software development cost estimation approachesâa survey. *Annals of software engineering*, 10(1-4):177–205, 2000.

[16] Brill, E. A simple rule-based part-of-speech tagger proceedings of the 3rd anlp san francisco.

[17] Britto, R., Freitas, V., Mendes, E., and Usman, M. Effort estimation in global software development: A systematic literature review. In *Global Software Engineering (ICGSE), 2014 IEEE 9th International Conference on*, pages 135–144. IEEE, 2014.

[18] Budgen, D., Turner, M., Brereton, P., and Kitchenham, B. Using mapping studies in software engineering. In *Proceedings of PPIG*, volume 8, pages 195–204. Lancaster University, 2008.

[19] Cerpa, N., Bardeen, M., Astudillo, C. A., and Verner, J. Evaluating different families of prediction methods for estimating software project outcomes. *Journal of Systems and Software*, 112:48–64, 2016.

[20] Chinthanet, B., Phannachitta, P., Kamei, Y., Leelaprute, P., Rungsawang, A., Ubayashi, N., and Matsumoto, K. A review and comparison of methods for determining the best analogies in analogy-based software effort estimation. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pages 1554–1557. ACM, 2016.

[21] Chiu, N.-H. and Huang, S.-J. The adjusted analogy-based software effort estimation based on similarity distances. *Journal of Systems and Software*, 80(4):628–640, 2007.

[22] Choetkiertikul, M., Dam, H. K., Tran, T., Pham, T., Ghose, A., and Menzies, T. A deep learning model for estimating story points. *arXiv preprint arXiv:1609.00489*, 2016.

[23] Choetkiertikul, M., Dam, H. K., Tran, T., Pham, T. T. M., Ghose, A., and Menzies, T. A deep learning model for estimating story points. *IEEE Transactions on Software Engineering*, 2018.

[24] Choi, P. S. P. S. V., Soonhwang. A rule-based approach for estimating software development cost using function point and goal and scenario based requirements. *Expert Systems with Applications*, 39(1):406–418, 2012.

[25] Cohn, M. *Agile estimating and planning*. Pearson Education, 2005.

[26] Costa, L. A. and Salvador, L. d. N. Ambiente de aprendizagem presencial e virtual integrados com a computação ubíqua: Um mapeamento sistemático da literatura. In *Memorias del XX Congresso Internacional de Informática Educativa, TISE*, volume 11, pages 211–220, 2015.

[27] Dragicevic, S., Celar, S., and Turic, M. Bayesian network model for task effort estimation in agile software development. *Journal of Systems and Software*, 127:109–119, 2017.

[28] El Bajta, M. Analogy-based software development effort estimation in global software development. In *Global Software Engineering Workshops (ICGSEW), 2015 IEEE 10th International Conference on*, pages 51–54. IEEE, 2015.

[29] Ezghari, S., Zahi, A., and Idri, A. A learning adaptation cases technique for fuzzy analogy-based software development effort estimation. In *Complex Systems (WCCS), 2014 Second World Conference on*, pages 492–497. IEEE, 2014.

[30] Garg, S. and Gupta, D. Pca based cost estimation model for agile software development projects. In *Industrial Engineering and Operations Management (IEOM), 2015 International Conference on*, pages 1–7. IEEE, 2015.

[31] Hamdy, A. Genetic fuzzy system for enhancing software estimation models. volume 4, pages 227–232, 2014.

[32] Hosni, M. and Idri, A. Software effort estimation using classical analogy ensembles based on random subspace. In *Proceedings of the Symposium on Applied Computing*, pages 1251–1258. ACM, 2017.

[33] Hosni, M., Idri, A., Nassif, A. B., and Abran, A. Heterogeneous ensembles for software development effort estimation. In *Soft Computing & Machine Intelligence (ISCMI), 2016 3rd International Conference on*, pages 174–178. IEEE, 2016.

[34] Huang, J., Li, Y.-F., Keung, J. W., Yu, Y. T., and Chan, W. An empirical analysis of three-stage data-preprocessing for analogy-based software effort estimation on the isbsg data. In *Software Quality, Reliability and Security (QRS), 2017 IEEE International Conference on*, pages 442–449. IEEE, 2017.

[35] Hussain, I., Kosseim, L., and Ormandjieva, O. Approximation of cosmic functional size to support early effort estimation in agile. *Data & Knowledge Engineering*, 85:2–14, 2013.

[36] Idri, A. and Abnane, I. Fuzzy analogy based effort estimation: An empirical comparative study. In *Computer and Information Technology (CIT), 2017 IEEE International Conference on*, pages 114–121. IEEE, 2017.

[37] Idri, A., azzahra Amazal, F., and Abran, A. Analogy-based software development effort estimation: A systematic mapping and review. *Information and Software Technology*, 58:206–230, 2015.

[38] Idri, A., Hosni, M., and Abran, A. Improved estimation of software development effort using classical and fuzzy analogy ensembles. *Applied Soft Computing*, 49:990–1019, 2016.

[39] Idri, A., Hosni, M., and Abran, A. Systematic literature review of ensemble effort estimation. *Journal of Systems and Software*, 118:151–175, 2016.

[40] Ionescu, V.-S. An approach to software development effort estimation using machine learning. In *Intelligent Computer Communication and Processing (ICCP), 2017 13th IEEE International Conference on*, pages 197–203. IEEE, 2017.

[41] Kazemifard, Z. A. N. M. A. M. F., M. Fuzzy emotional cocomo ii software cost estimation (fecsce) using multi-agent systems. *Applied Software Computing Journal*, 11(12):2260–2270, 2011.

[42] Keele, S. Guidelines for performing systematic literature reviews in software engineering. In *Technical report, Ver. 2.3 EBSE Technical Report. EBSE*. sn, 2007.

[43] Khatibi, E. and Bardsiri, V. K. Model to estimate the software development effort based on in-depth analysis of project attributes. *IET Software*, 9(4):109–118, 2015.

[44] Klein, D. and Manning, C. D. Accurate unlexicalized parsing. In *Proceedings of the 41st annual meeting of the association for computational linguistics*, 2003.

[45] Kocaguneli, E., Gay, G., Menzies, T., Yang, Y., and Keung, J. W. When to use data from other projects for effort estimation. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*, pages 321–324. ACM, 2010.

[46] Kocaguneli, E., Menzies, T., Bener, A., and Keung, J. W. Exploiting the essential assumptions of analogy-based effort estimation. *IEEE Transactions on Software Engineering*, 38(2):425–438, 2012.

[47] Li, J., Ruhe, G., Al-Emran, A., and Richter, M. M. A flexible method for software effort estimation by analogy. *Empirical Software Engineering*, 12(1):65–106, 2007.

[48] Manapian, A. and Prompoon, N. Software time estimation model for requirements change based

on software prototype profiles using an analogy estimation method. In *Computer Science and Engineering Conference (ICSEC), 2014 International*, pages 366–371. IEEE, 2014.

[49] Manikavelan, D. and Ponnusamy, R. Software cost estimation by analogy using feed forward neural network. In *Information Communication and Embedded Systems (ICICES), 2014 International Conference on*, pages 1–5. IEEE, 2014.

[50] Menzies, T., Chen, Z., Hihn, J., and Lum, K. Selecting best practices for effort estimation. *IEEE Transactions on Software Engineering*, 32(11), 2006.

[51] Moharreri, K., Sapre, A. V., Ramanathan, J., and Ramnath, R. Cost-effective supervised learning models for software effort estimation in agile environments. In *Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual*, volume 2, pages 135–140. IEEE, 2016.

[52] Moosavi, S. H. S. and Bardsiri, V. K. Satin bowerbird optimizer: A new optimization algorithm to optimize anfis for software development effort estimation. *Engineering Applications of Artificial Intelligence*, 60:1–15, 2017.

[53] Najadat, A. I. S. Y., H. Predicting software projects cost estimation based on mining historical data. *ISRN Software Engineering*, pages 1–8, 2012.

[54] Nanda, S., Soewito, B., et al. Modeling software effort estimation using hybrid pso-anfis. In *Intelligent Technology and Its Applications (ISITIA), 2016 International Seminar on*, pages 219–224. IEEE, 2016.

[55] Nassif, A. B., Capretz, L. F., and Ho, D. Analyzing the non-functional requirements in the desharnais dataset for software effort estimation. *arXiv preprint arXiv:1405.1131*, 2014.

[56] Ochodek, M. Functional size approximation based on use-case names. *Information and Software Technology*, 80:73–88, 2016.

[57] Panda, A., Satapathy, S. M., and Rath, S. K. Empirical validation of neural network models for agile software effort estimation based on story points. *Procedia Computer Science*, 57:772–781, 2015.

[58] Petersen, K., Vakkalanka, S., and Kuzniarz, L. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1–18, 2015.

[59] Phannachitta, P., Keung, J., Monden, A., and Matsumoto, K. A stability assessment of solution adaptation techniques for analogy-based software effort estimation. *Empirical Software Engineering*, pages 1–31, 2016.

[60] Rao, P. S., Reddi, K. K., and Rani, R. U. Optimization of neural network for software effort estimation. In *Algorithms, Methodology, Models and Applications in Emerging Technologies (ICAMMAET), 2017 International Conference on*, pages 1–7. IEEE, 2017.

[61] Ren, X., Dai, Y., and Zhou, L. Application in effort estimation of collaborative filtering. In *Computational Intelligence and Design (ISCID), 2013 Sixth International Symposium on*, volume 1, pages 330–333. IEEE, 2013.

[62] Resmi, V., Vijayalakshmi, S., and Chandrabose, R. S. An effective software project effort estimation system using optimal firefly algorithm. *Cluster Computing*, pages 1–10, 2017.

[63] Rijwani, P. and Jain, S. Enhanced software effort estimation using multi layered feed forward artificial neural network technique. *Procedia Computer Science*, 89:307–312, 2016.

[64] Rumjaun, S. D. N. S. S., Gutteea, K. A., and Nagowah, L. Effortest-an enhanced software effort estimation by analogy method. *ADBU Journal of Engineering Technology*, 5(2), 2016.

[65] Sánchez, R. and Pinto-Roa, D. P. Design of software effort estimation models an approach based on linear genetic programming. In *Computer Conference (CLEI), 2017 XLIII Latin American*, pages 1–10. IEEE, 2017.

[66] Satapathy, S. M. and Rath, S. K. Empirical assessment of machine learning models for effort estimation of web-based applications. In *Proceedings of the 10th Innovations in Software Engineering Conference*, pages 74–84. ACM, 2017.

[67] Shepperd, M. Software project economics: a roadmap. In *2007 Future of Software Engineering*, pages 304–315. IEEE Computer Society, 2007.

[68] Shepperd, M. and Kadoda, G. Comparing software prediction techniques using simulation. *IEEE Transactions on Software Engineering*, 27(11):1014–1022, 2001.

[69] Shivhare, J. and Rath, S. K. Software effort estimation using machine learning techniques. In *Proceedings of the 7th India Software Engineering Conference*, page 19. ACM, 2014.

[70] Silhavy, R., Prokopová, Z., and Silhavy, P. Algorithmic optimization method for effort estimation. *Programming and Computer Software*, 42(3):161–166, 2016.

[71] Singh, S. P. and Kumar, A. Software cost estimation using homeostasis mutation based differential evolution. In *Intelligent Systems and Control (ISCO), 2017 11th International Conference on*, pages 173–181. IEEE, 2017.

[72] Sternberg, R. J. Component processes in analogical reasoning. *Psychological Review*, 84:353–378, 1977.

[73] Thamarai, I. and Murugavalli, S. Using differential evolution in the prediction of software effort. In *Advanced Computing (ICoAC), 2012 Fourth International Conference on*, pages 1–3. IEEE, 2012.

[74] Thamarai, I. and Murugavalli, S. An evolutionary computation approach for project selection in analogy based software effort estimation. *Indian Journal of Science and Technology*, 9(21), 2016.

[75] The Standish Group International. The chaos report, 1995.

[76] Usman, M., Mendes, E., Weidt, F., and Britto, R. Effort estimation in agile software development: a systematic literature review. In *Proceedings of the 10th International Conference on Predictive Models in Software Engineering*, pages 82–91. ACM, 2014.

[77] Velarde, H., Santiesteban, C., Garcia, A., and Casillas, J. Software development effort estimation based-on multiple classifier system and lines of code. *IEEE Latin America Transactions*, 14(8):3907–3913, 2016.

[78] Wen, J., Li, S., Lin, Z., Hu, Y., and Huang, C. Systematic literature review of machine learning based software development effort estimation models. *Information and Software Technology*, 54(1):41–59, 2012.

[79] Wu, D., Li, J., and Bao, C. Case-based reasoning with optimized weight derived by particle swarm optimization for software effort estimation. *Soft Computing*, pages 1–12, 2017.

[80] Zare, F., Zare, H. K., and Fallahnezhad, M. S. Software effort estimation based on the optimal bayesian belief network. *Applied Soft Computing*, 49:968–980, 2016.

[81] Zhang, C., Tong, S., Mo, W., Zhou, Y., Xia, Y., and Shen, B. Esse: an early software size estimation method based on auto-extracted requirements features. In *Proceedings of the 8th Asia-Pacific Symposium on Internetware*, pages 112–115. ACM, 2016.

[82] Zhang, W., Yang, Y., and Wang, Q. Using bayesian regression and em algorithm with missing handling for software effort prediction. *Information and software technology*, 58:58–70, 2015.